Brock University

Department of Computer Science

Evolutionary Synthesis of Stochastic Gene Network Models using Feature-based Search Spaces

Janine Imada
Technical Report # CS-09-10
November 2009

Brock University
Department of Computer Science
St. Catharines, Ontario
Canada L2S 3A1
www.cosc.brocku.ca

Evolutionary Synthesis of Stochastic Gene Network Models using Feature-based Search Spaces

Janine Imada

Department of Computer Science

Submitted in partial fulfillment
of the requirements for the degree of

Master of Science

Faculty of Mathematics and Science, Brock University
St. Catharines, Ontario

Abstract

A feature-based fitness function is applied in a genetic programming system to synthesize stochastic gene regulatory network models whose behaviour is defined by a time course of protein expression levels. Typically, when targeting time series data, the fitness function is based on a sum-of-errors involving the values of the fluctuating signal. While this approach is successful in many instances, its performance can deteriorate in the presence of noise. This thesis explores a fitness measure determined from a set of statistical features characterizing the time series' sequence of values, rather than the actual values themselves. Through a series of experiments involving symbolic regression with added noise and gene regulatory network models based on the stochastic π-calculus, it is shown to successfully target oscillating and non-oscillating signals. This practical and versatile fitness function offers an alternate approach, worthy of consideration for use in algorithms that evaluate noisy or stochastic behaviour.

Acknowledgments

I would like to thank the following individuals and organizations for their support in preparing this thesis:

- B. Ross for superb supervision
- B. Ombuki-Berman and V. Wojcik for participating on my supervisory committee
- C. Fairchild and T. Whitehead (SHARCNET) for technical assistance
- NSERC for generous funding
- SHARCNET for use of their HPC facilities
- S. Burnside for help on the home front

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Gene regulatory networks (GRNs) are complex biological systems governing gene expression which serve to control important cellular processes. Considerable research efforts have been put forth in recent years to model and learn the dynamic behaviour of these networks. The ability to automatically construct GRN models provides biologists with a tool for discovery and insight.

Stochasticity has been recognized as an influential element in these systems due to the small number of molecules involved. A stochastic model based on gene gates which represent biological interactions as expressed by the stochastic π-calculus has been developed. The modularity of this model points to genetic programming (GP) as a favourable algorithm for model inference. However, effective evaluation of candidate GP programs can be hindered by the stochastic element present in the model. The standard approach, which is based on a sum-of-errors between the target behaviour and actual time course values of the candidate expression, can suffer in performance due to the presence of noise. There is a need to develop effective methods to measure fitness when dealing with noisy and stochastic behaviour.

This thesis explores an alternate fitness function which is based on characterizing GRN behaviour by a set of statistical features. The use of features is a practical approach which can be easily tailored to suit a variety of behaviours. For each problem at hand, a subset of features is chosen from a larger, comprehensive set of time series features, and incorporated into a fitness function which determines a sum-of-errors between the subset and corresponding targeted feature values to evaluate a candidate expression's behaviour.

Through a series of experiments involving symbolic regression and gene gate models with varying complexity, the effectiveness and versatility of this feature-based fitness function is demonstrated by considering both oscillating and non-oscillating systems. In light of the positive results obtained in these experiments, it is suggested that the feature-based

fitness function is worthy of consideration for use in algorithms which deal with noisy or stochastic behaviours.

Subsequent sections are laid out as follows. Background information regarding genetic programming, gene regulatory network models and the particular gene gate model inferred in this study is found in Chapter 2. Following this is a review of related work (Chapter 3), and then in context of this review, the problem tackled in this thesis is presented in Chapter 4. Chapter 5 provides details on the feature-based fitness function, followed by 2 sections which document the series of GP experiments which were conducted to explore the effectiveness of the fitness function. Chapter 6 covers the symbolic regression experiments, while Chapter 7 presents those inferring the gene gate models. Following this is a section discussing the combined results from all of the experiments (Chapter 8), and Chapter 9 states the conclusions and suggests further work.

It should be noted that a portion of the following work was previously documented in a paper prepared for GECCO 2008 [28].

Chapter 2

Background

2.1 Genetic Programming

Genetic Programming (GP) is an evolutionary computational algorithm which offers a framework to automatically synthesize programs aimed to produce a targeted behaviour [35] [50]. Programs are expressed in the form of trees, constructed from a set of specified building blocks consisting of functions and terminals.

2.1.1 Context within Artificial Intelligence and Machine Learning

GP is a subset of Machine Learning, which is in turn a subset of Artificial Intelligence. Artificial Intelligence is a broad field which addresses "the mechanisms underlying thought and intelligent behavior and their embodiment in machines" [1]. Among the many topics associated with Artificial Intelligence is Machine Learning which deals with systems that improve their performance by learning through experience. Within Machine Learning is the field of evolutionary computation, a set of biologically-inspired stochastic algorithms, which tackle search and combinatorial optimization problems. GP is one such algorithm.

Genetic Programing is capable of building programs of variable length and structure. Its tree construct allows for nesting of modular components, and readily accommodates typing constraints or adherence to a grammar. Since the algorithm involves a random element, solutions generated by a run are not guaranteed to be optimal. However, this may be regarded as a positive feature when dealing with real world problems in that it can be used as a tool for discovery of relevant relationships or configurations, not always evident through observation.

2.1.2 GP Algorithm

In the GP algorithm (Figure 2.1), a population of programs is maintained. Each program, which is also referred to as an individual, expression or candidate solution, is represented by a tree. The tree consists of internal nodes selected from a set of basic functions and leaf nodes selected from a set of terminals. Genetic operators such as crossover and mutation are applied to selected individuals from the population of trees, creating offspring which vary from their parents to make up the next generation. Since selection favours those which score better fitness values, the population progressively evolves through a series of generations to more closely behave like the target. The GP algorithm eventually converges such that individuals in subsequent generations do not register any improvements in performance. The algorithm is considered stochastic in that the operators (crossover, mutation and selection) and population initialization contain steps in which direction is chosen with a probability.

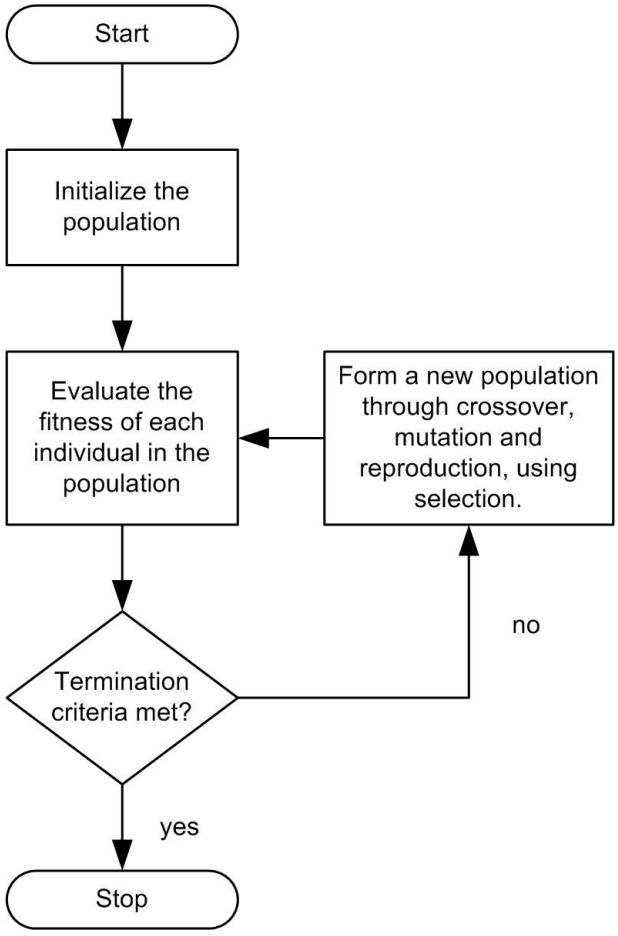


Figure 2.1: GP Algorithm

2.1.3 GP System Design

There are several elements of a GP system which must be tailored to suit the problem at hand.

Program Representation

The function and terminal sets define the building blocks of the expression and provide guidance on how they are to be pieced together in the program tree. The sets considered in this thesis were either single or strongly typed. Single type sets involve functions whose arguments and results have the same type. Strongly typed sets contain functions with arguments and result values with a mixture of types, and tree construction is constrained such that they correctly match the types.

All functions must ensure closure, meaning that a result must be able to be calculated for every possible input. Functions are deemed "protected" if some provisions have been incorporated to ensure closure. For instance, division often needs to be protected to avoid divide by zero errors.

Target Behaviour

The behaviour which is being targeted and searched for needs to identiifed.

Fitness Function

The fitness function quantifies how well the expression performs in meeting the target behaviour.

2.1.4 GP Parameters and Settings

In addition to these design elements of the GP, there are several parameters and settings to consider. In this section, parameters as applied to the GP runs contained in this thesis are described. More detailed explanations of these parameters are provided in [35] and [50].

Population Size

The population size refers to the constant number of candidate solutions in the population which is maintained from generation to generation.

Maximum Number of Generations

This parameter is one way in which termination of the algorithm is specified. In this thesis, all GP runs started from generation 0 (the initial population) and continued up until the maximum number of generations was reached.

Probability of Crossover

This is the probability that a selected individual will be subject to crossover. During crossover, a randomly selected node (and its subtree of which it is the root) from one parent is swapped with a randomly selected node (and its subtree) from a second parent. Crossover must respect tree depth restrictions and type compatibility.

Probability of Standard Mutation

This is the probability that an individual in the population will be subject to standard mutation, in which a randomly selected node (and its subtree) is replaced with a randomly constructed branch. This branch must be no longer than the maximum regenerative depth and type compatible.

Probability of Shrink Mutation

This the probability that an individual in the population will be subject to shrink mutation, in which a randomly selected node is replaced by one of its child nodes of compatible type.

Probability of Reproduction

This is the probability that a selected individual will be carried over to the next generation untouched.

Elitism

Elitism is when the individuals scoring the best fitness are carried over to the next generation.

Selection

Selection is the operator which selects individuals for reproduction or crossover. Common approaches for selection are roulette wheel and tournament selection. In this thesis, tournament selection is used in which n individuals are chosen randomly from the population and

from this sub-group, the individual with the best fitness is selected. n is referred to as the tournament size. Tournament selection is an effective and efficient rank-based approach.

Initial Population

This parameter describes how the population is initialized for generation 0. Popular approaches are full, grow and ramped half and half. The full method randomly generates initial trees in which the depth of all leaf nodes is equal to the maximum depth. The grow method randomly generates initial trees which meet the initial tree depth restrictions, producing a population composed of trees with variable depth. The ramped half and half method combines the full and grow approaches. Here, the initial population is composed of trees with depths ranging from the minimum to the maximum initial depths in equal proportion. For each of these depths, half of the trees are full and half are generated using the grow method. In this report, both grow and ramped half and half approaches are used, depending on the experiment.

Minimum Initial Tree Depth

This parameter refers to the minimum tree depth permitted in the initial population.

Maximum Initial Tree Depth

This parameter refers to the maximum tree depth permitted in the initial population.

Maximum Tree Depth

This parameter restricts the maximum tree depth of any individual throughout the entire GP run.

Probability that Crossover Point is a Branch

This parameter is used during the crossover operation. It prescribes the probability that the randomly selected node is an internal node as opposed to a leaf node.

Maximum Regenerative Depth for Mutation

The maximum regenerative depth for mutation is the maximum depth of a subtree that can replace the selected node in standard mutation.

Maximum Number of Retries

During a GP run, several attempts may be necessary to successfully perform a genetic operation in the presence of restrictions such as maximum tree depths. This parameter limits the number of times that a genetic operation can be attempted. It serves to increase the success rate of genetic operations, while avoiding infinite loops.

2.2 Gene Regulatory Networks

Gene regulatory networks describe cellular interactions involving DNA, RNA transcription and protein synthesis. Currently, modelling and learning these networks is a large area of study. Two recent surveys describing ways in which bionetworks, including gene networks, are modelled have been provided by Tkačik & Bialek [63], and Fisher & Henzinger [19]. Several of the approaches outlined in these reviews include:

- **Reaction Rate Equations**
  Linear or nonlinear differential equations model reactions between biomolecules (e.g. proteins, genes) in the system and the rates at which these reactions occur. This model is deterministic. The S-system model fits into this category. Stochasticity can be introduced through the use of stochastic differential equations.

- **Boolean Networks**
  In this deterministic model, nodes represent biomolecules which are either active ("1") or inactive ("0"). As the model is executed, the subsequent state of a node is determined by the current state of the connecting nodes. Probabilistic versions of this model exist.

- **Bayesian Networks**
  This model is a directed, acyclic graph where nodes represent biological variables of interest and edges signify dependencies which are quantified by tables of conditional probabilities associated with each node. There are dynamic versions of this model.

- **Petri Nets**
  Petri nets are graphs with two types of nodes representing places (biomolecules) and transitions (events) connected together by edges. Tokens (representing a signal or quantity) can move concurrently along edges from place to place. There are stochastic versions of Petri nets.

- **Process Calculi**
  Processes associated with biomolecules are elements in this model. Execution of the model produces a sequence of events. During events, processes communicate which corresponds to an interaction between the molecules. This non-deterministic model becomes stochastic with the addition of reaction rates.

The gene regulatory network model focused on in this thesis is an abstract model composed of modular gene gates based on a stochastic process calculus. This model is further described in the following section.

2.3 Stochastic Gene Gate Model

The gene regulatory network model targeted in this research is based on recent work by Blossey, Cardelli & Phillips [7]. They have developed a modular approach built upon elements called "gene gates". This model can be described as:

- **Stochastic**
  Recognizing that the presence of noise and stochasticity are essential in gene networks, the gates are composed of stochastic π-calculus expressions.

- **Abstract**
  Intermediary biological steps are omitted.

- **Modular**
  Biological detail can be added to the definition of the gate without changing the topology of the network.

- **Computational**
  This is an executable model as opposed to a mathematical one. Upon execution, this model yields a sequence of events with causal relationships [19].

- **Dynamic**
  Execution of this model results in a time course of gene expression levels.

Gene gates model the basic regulatory mechanism which involves the production of proteins (translation) from DNA through the production of RNA (transcription). In this model, transcription and translation are considered a single action. Further interactions and actions incorporated in the model include repression, activation, degradation and stochastic delay [10][42].

In a subsequent publication, Blossey et al. expanded their model to include more biological detail such as repressor dimerization and tetramerization. As well, transcription and translation were treated as separate operations [8]. Note that these enhancements were not included in the model used in this study.

2.3.1 The Stochastic π-calculus

The underlying language of the gene gate model is the stochastic π-calculus. The π-calculus is a process algebra capable of modelling concurrent systems in a compact manner. Since it is defined by a formal language, π-calculus constructs can be pieced together in the form of a program.

Basic elements of the π-calculus are communication channels (receiving ($?$) and sending ($!$)). A matching set of complementary channels allows processes to interact and communicate. Once an interaction takes place, the process changes to its next state which is specified after the dot, ".". In the gene gate model, these interactions are simple in that they serve as signals, without any exchange of data. Processes can be executed in parallel ($|$) or be subject to choice ($+$) among alternate processes.

The stochastic element in the stochastic π-calculus is achieved through the inclusion of rates, which enable the channels to be quantified. In the gene gate model, rates are expressed as communication rates for each channel and as stochastic delays (τ). Higher rates lead to shorter delays on average. Once a stochastic π-calculus program is formed, it can undergo a probabilistic simulation based on the Gillespie algorithm, producing a time series measuring the dynamic quantity of a channel over time.

2.3.2 Gene Gates and Other Network Elements

Gene gates are modular constructs which when combined in parallel create the gene regulatory network model. Depending on their complexity, they can be parameterized by elements such as interaction sites, rates and transcription factors. A description of the gates and other network elements used in the GP experiments are provided below. The schematic diagrams for the gates and networks were taken from [7], and they follow the notation described in Figure 2.2 [10].

Transcription Factor, $tr(b)$

A transcription factor is a protein that can regulate (inhibit or stimulate) transcription (RNA synthesis). This particular network element offers two possible behaviours: It either (a)

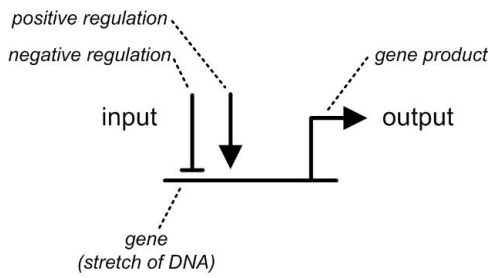


Figure 2.2: Notation for Gene Gate and Network Diagrams

produces a protein that binds to a receiving site $?b$, or (b) degrades ("0") following a stochastic delay, τ_δ. If the protein binds to a site, it returns to its initial state, $tr(b)$, and is available for a subsequent interaction. According to process algebra, once one of these behaviours is realized, the other option is discarded.

$$tr(b) = !b.tr(b) \quad \text{(a)}$$
$$+ \tau_\delta.0 \quad \text{(b)}$$

where δ is the degradation rate

Repressible Transcription Factor, $rtr(b, r)$

This network element provides a transcription factor b, that can be repressed through site r. This element offers three possible behaviours: It either (a) produces a protein that can bind to a receiving site $?b$, or (b) be repressed by binding to a receiving site $?r$, or (c) degrades ("0") following a stochastic delay, τ_δ. If the protein binds to a site, it subsequently returns to its initial state, $rtr(b, r)$. However, if the behaviour executed is repression, the element subsequently degrades.

$$rtr(b, r) = !b.rtr(b, r) \quad \text{(a)}$$
$$+ !r.0 \quad \text{(b)}$$
$$+ \ \tau_\delta.0 \quad \text{(c)}$$

where δ is the degradation rate

Repressor, $rep(r)$

This element offers continuous quantities of repressor receiving site r, which if bound to, prevents production of the repressible transcription factor, $rtr(b, r)$.

$$rep(r) = ?r.rep(r)$$

Simple Negative Regulation, Neg Gate

The Neg gate, $neg(a, b)$, produces negative regulation such that in the presence of transcription factor a, the production of gene product b is inhibited (Figure 2.3). This gate offers two possible behaviours: It either (a) has a transcription factor bind to its promoter site a which effectively inhibits transcription, or (b) provides transcription, producing factor $tr(b)$. If the gate is inhibited, it returns to its initial state, $neg(a, b)$, following a stochastic delay, τ_η. If transcription occurs, the gate returns to its initial state, $neg(a, b)$, and is available for subsequent interaction along with product $tr(b)$.

$$neg(a, b) = ?a.\tau_\eta.neg(a, b) \quad \text{(a)}$$
$$+ \tau_\epsilon.\,(tr(b)|neg(a, b)) \quad \text{(b)}$$

where η is the inhibition rate

ϵ is the production (constitutive translation) rate

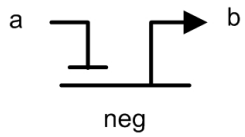


Figure 2.3: Neg Gate

Neg Gate with Parametrized Product, Negp Gate

In order to construct more sophisticated combinatorial networks, a gate with more flexible parameters was created. This gene gate, $negp(a, (\epsilon, \eta), p)$, takes on additional rate parameters and specifies a more generalized product, p (Figure 2.4). In the networks that

are considered in this thesis, gene product p, can be either of the two transcription factors, $tr(b)$ or $rtr(b, r)$. The Negp gate is defined as follows:

$$negp\,(a, (\epsilon, \eta)\,, p) = ?a.\tau_{\eta}.negp\,(a, (\epsilon, \eta)\,, p) + \tau_{\epsilon}.\,(p\,()\,|negp\,(a, (\epsilon, \eta)\,, p))$$

where η is the inhibition rate

ϵ is the production (constitutive translation) rate

$p\,()$ is product generation

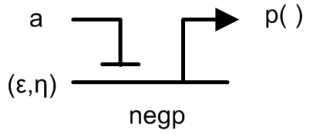


Figure 2.4: Negp Gate

With this definition, $neg(a, b)$ is a special case of $negp(a, (\epsilon, \eta)\,, p)$, where the rates, (ϵ, η), are taken out of parameter list and p is set to transcription factor $tr(b)$.

2.3.3 Gene Gate Expressions

Gene gates are combined in parallel to produce expressions which model the gene regulatory network. The gene product from one gate can self-regulate or serve as a regulator to other gates, forming complicated relationships and interactions. This, in conjunction with the stochastic rates, make it very difficult to predict the behaviour of an expression. To illustrate how gene gates can interact to produce regulatory circuits, two simple networks which are frequently referred to in literature are presented below.

Bistable Network: $neg(a, b)|neg(b, a)$

A bistable network composed of two proteins, a and b, can operate in one of two stable states where one channel has a high level of expression, while the other remains low. Switching between the two states, in which the channels concurrently flip their levels of expression, is triggered by either external inputs, such as a pulse of additional gene product, or by internal noise.

Gene gates can be combined to produce bistable behaviour with intrinsic switching [7]. The network is illustrated in Figure 2.5.

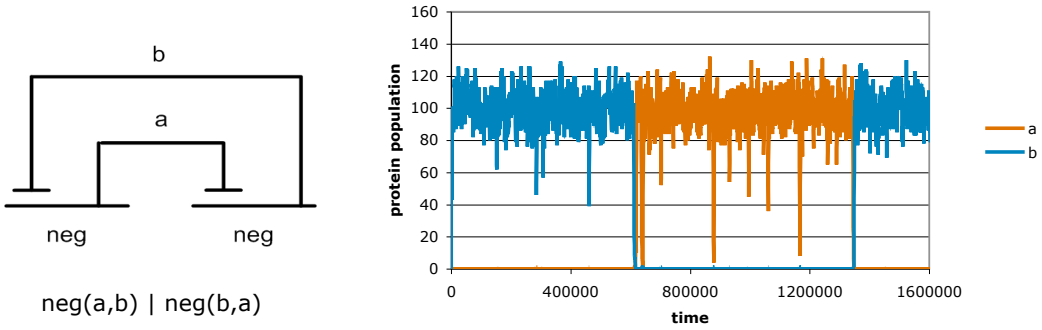


Figure 2.5: Bistable Network

Here, each neg gate produces one of the proteins, while the other protein serves to retard it production. At the start of the simulation, due to stochastic effects, one of the proteins dominates, keeping the population of the other product low. However, at some point, the balance is stochastically tipped, resulting in a switch in the levels of protein expression and the other stable state is established. Because switching is triggered stochastically, the duration of each stable state is highly irregular.

Repressilator: $neg(a,b)|neg(b,c)|neg(c,a)$

The repressilator is an artificial circuit [17], composed of three proteins with oscillating levels of expression, each peaking in sequence. A network consisting of 3 gene gates can describe this behaviour [7] (Figure 2.6).

In this circuit, when lots of protein b is being produced, the production of protein c is reduced, thus allowing more protein a to be created. Increased quantities of a shuts down the production of b, which leads to an increase in c, and so on. This cascading effect results in alternating cyclic behaviour. Different combinations of rates have been studied and were found to affect the regularity and uniformity of the cycles [8].

2.3.4 Expression Simulation

Once a gene gate expression is pieced together, it can be simulated to produce a time course tracking the change in gene product quantities. A simulator for the stochastic π-calculus called the Stochastic Pi Machine (SPiM) is available to execute gene gate expressions [49]. This simulation is based on the Gillespie algorithm [24] which is a Monte Carlo procedure to stochastically simulate a system of chemical reactions.

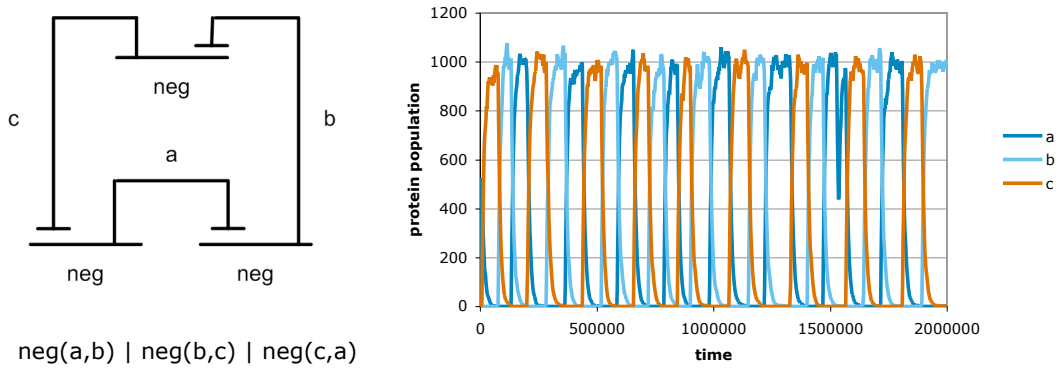


Figure 2.6: Repressilator Network

During the simulation of an expression, SPiM determines the set of possible reactions among all processes which are operating in parallel. The possible reactions are made up of delays and matching pairs of sending and receiving channels. There is a probability associated with each of these reactions as defined by their corresponding rates. The next reaction and associated time increment are then chosen stochastically according to this set of reactions and their probabilities. By repeating this procedure and recording the number of output sites, (!), for each channel at each step, a time course of protein (gene product) population levels is produced.

A more thorough discussion of the stochastic π-calculus and its simulation via SPiM is found in the supplementary material associated with [8].

Chapter 3

Related Work

Learning stochastic gene regulatory network models with genetic programming using a feature-based fitness function is associated with several broad fields of study:

1. Learning gene regulatory network models
2. Learning dynamic models
3. Feature-based search spaces

Subsequent sections in this chapter address each of these fields, first providing context of the thesis topic within the field, followed by a discussion of the most relevant, related work.

3.1 Learning Gene Regulatory Network Models

3.1.1 Context

Machine learning methods are used extensively in bioinformatics. In a recent survey of the field, Larrañaga et al. [37] sorted machine learning applications into 7 biological domains. Within this classification system, the inference of gene regulatory networks was placed in the intersection between the systems biology and microarray categories.

In general, GRN models can be constructed to serve 2 purposes [63]. Firstly, they can provide a topological model of the system, identifying regulatory relationships between biomolecules of interest (e.g. genes, proteins). An example where machine learning has been applied to infer these static models is provided by Supper et al. [61] in which 4 different methods (Bayesian networks, multiple linear regression, CART decision trees, support vector machines) were applied to infer regulatory dependencies between genes

from microarray data. A second purpose of GRN models is to simulate the dynamics of the network, namely the change in gene expression levels over time. It is this particular type of model which is being considered in this thesis.

Real temporal gene expression data ("in vivo") is obtained from microarray experiments. Microarray data is characterized by a limited number of samples covering a short duration for many genes. This data can be noisy and have missing values. Consequently, the nature of this data poses a computational challenge when inferring temporal models [6]. As such, it is common practice in current research for artificially-derived data from simulations ("in silico") to be used to learn GRN models.

Many different algorithms have been applied extensively to learn numerous types of temporal models. Evolutionary algorithms have been identified as a noteworthy machine learning tool for the optimization of gene networks and other bionetworks [37].

The next three sections highlight work that has made use of evolutionary algorithms to infer deterministic and probabilistic models. Following this is a section which examines the fitness functions used in each of these studies.

3.1.2 Learning Deterministic GRN Models with Evolutionary Algorithms

To carry out the task of learning GRN models, evolutionary algorithms evolve a population of candidate models which are interpreted in order to obtain a time series upon which the fitness is evaluated. Deterministic models are those which generate exactly the same time course of values each time the model is simulated.

A sampling of papers which use evolutionary algorithms (other than GP) to evolve temporal GRN models are detailed below:

- Kitagawa & Iba [33] used an evolutionary algorithm to infer functional Petri nets modelling metabolic pathways from artificial data.
- Kikuchi et al. [32] used a genetic algorithm to determine the parameters for S-system GRN models.
- Jin & Sendhoff [30] used Evolution Strategies to evolve the parameters for GRN models composed of differential equations, targeting bistable and oscillating behaviours.
- François & Hakim [22] used an evolutionary algorithm to infer a set of differential equations to model GRNs, targeting bistable and oscillating behaviours.

Among evolutionary algorithms, GP is widely used to evolve both the structure and parameters of temporal gene networks and other bionetworks containing similar mechanisms. The following papers involve work that have used GP to evolve deterministic GRN models:

- Cho et al. [11] evolved GRNs and other biochemical networks using an S-tree model which describes a sparse network of non-linear differential equations.
- Koza et al. [36] evolved a network of chemical reactions (including reaction rates) describing a metabolic pathway by simulating the network as an analog electric circuit model.
- Streichert et al. [59] used differential equations to evolve the topology and model for a GRN.
- Ando et al. [3] and Sakamoto & Iba [56] evolved differential equations to model a GRN. GP was used to optimize the structure of the network in conjunction with the LMS (least mean square) method which served to refine the parameters.

3.1.3 Learning Probabilistic GRN Models with Evolutionary Algorithms

Stochasticity has been recognized as an influential element in GRNs because of the small number of molecules involved [18]. Learning of probabilistic GRN models has been the subject of several recent studies, particularly to evolve oscillating, switching or bistable behaviours. Probabilistic models are subject to stochastic variation during interpretation resulting in time courses of gene expression levels which differ each time they are simulated. The stochastic π-calculus gene gate models constructed in this thesis fit into this category.

The following papers involve the learning of probabilistic GRN models through evolutionary algorithms:

- In two separate papers, Leier et al. [41] and Leier & Burrage [40] used the Gillespie algorithm to stochastically simulate a set of elementary reactions using set-based GP, targeting oscillating [41] and switching [40] behaviours. Both papers commented on the effect of stochasticity in their models. Oscillation observed in two examined networks was attributed to the stochastic element, because the corresponding deterministic models, consisting of ordinary differential equations, did not exhibit similar cyclic behaviour. As well, for the two highlighted networks which behaved as switches, shifting between high and low levels of expression was attributed to

the inherent noise in the system, since the equivalent deterministic model required external injections to trigger the switching.

- Chu [12] used the Gibson-Bruck algorithm to simulate a set of reactions and rates obtained from an evolutionary algorithm, targeting oscillating behaviour.

- Qian et al. [51] used GP in combination with Kalman filtering (to estimate parameters) to evolve differential equation models for GRNs . Within the model, terms accounting for intrinsic and external Gaussian noise were added.

- With focus on studying the evolution of development, Drennan & Beer [16] used a genetic algorithm to evolve stochastic models. The candidate models resembled snippets of DNA, with bases representing genes and a set of promoter, enhancer and repressor sites. Stochastic simulation was performed through an algorithm aimed to minimize free energy [15].

3.1.4 Learning Deterministic GRN Models with Evolutionary Algorithms Amid Added Noise

In the following studies, noise was added to experiments involving deterministic models for various reasons:

- In several of the evolutionary algorithm papers cited in Section 3.1.2 , noise was added to the target data to test the robustness of the approach when exposed to real-world noisy data [3] [11] [56].

- Knabe et al. [34] introduced noise into the input of the network. The motivation was to examine the effects of noisy, external stimuli on the evolution of periodic behaviour.

3.1.5 Fitness Functions used to Learn GRN Models

In all of the evolutionary algorithm papers cited above, other than the ones dealing with oscillating or switching behaviours, the approach to fitness evaluation was based on the traditional sum-of-errors (absolute or squared and/or normalized) from the targeted time series values. In some cases, parsimony was encouraged through the addition of a size penalty [3][33][56] or a term encouraging sparse networks [32].

A summary of the approaches taken in those papers which departed from the standard sum-of-errors is provided below:

- To target oscillation, Chu [12] based the fitness function on autocorrelation, thus focussing on matching a specific periodicity and tolerating variation in phase and amplitude.
- Drennan & Beer [16] looked for repressilator behaviour by counting the number of out-of-phase cycles exhibited by 3 or more proteins.
- François & Hakim [22] targeted both bistable switching and oscillating behaviour. For the bistable switch, a sum-of-squared error from prescribed concentrations was used along with a size penalty. External pulses were applied to incur switching. For the oscillating behaviour, fitness was based on differences from specified concentration levels sampled at half-period intervals, implying that a specific amplitude, phase and frequency were targeted.
- Leier & Burrage [40] established a set of constraints to identify switching behaviour based on exceeding high levels and falling below low levels for minimum durations, along with a constraint on the time to switch between levels.
- Leier et al. [41] targeted oscillating behaviour by using a formula based on a Fast Fourier Transform, that rewarded sustained oscillation. Fitness for each candidate network was obtained by averaging the fitness over several (20) simulations.

3.2 Learning Dynamic Models

3.2.1 Context

The learning of GRN models to produce temporal behaviour can be considered a subset of the broader field of learning dynamic systems (systems whose signals change over time). Activity in this field is often related to financial forecasting and modelling of noisy or chaotic systems, and learning has been performed with real life data, specifically geared to accommodate noise. This section focuses on the learning of dynamic models through the use of evolutionary algorithms, with particular attention paid to GP.

3.2.2 Learning Probabilistic Dynamic Models with Evolutionary Algorithms

A single paper was identified which dealt with learning a probabilistic model through evolutionary algorithms. Ross [55] used grammar-guided GP to evolve stochastic π-calculus expressions, targeted to generate certain monotonic behaviours.

3.2.3 Learning Noisy Dynamic Models with Evolutionary Algorithms

Learning time series which contain noise, either added or inherent in the target behaviour, has been the focus of numerous evolutionary algorithm studies. The following are a sample of papers with special emphasis on works involving GP and/or features:

- Borrelli et al. [9] used GP with noise added to the target behaviour to develop models for financial forecasting purposes. A multi-objective fitness function was used in which one of the objectives was based on the sum-of-squared error from the time series data, and the second objective based on a combination of sum-of-errors for two statistical features. It was found that this multi-objective approach resulted in improved performance.

- Rodriguez-Vazquez & Flemming [53] and Rodriguez et al. [54] used GP to evolve non-linear NARMAX models to describe oscillating chaotic systems [53] and dynamic systems [54]. Again, a multi-objective approach was taken, incorporating model complexity, performance and model criteria.

- Schwaerzel & Bylander [57] used GP to predict currency exchange rates. The function set included statistical features parameterized by length and lag. The fitness function was based on the sum-of-squared error from the time series data.

- Hinchliffe & Willis [27] modelled dynamic systems using GP based on the NARX model. Both single and multi-objective experiments were carried out. The single objective was based on the standard sum-of-squared error approach. The multi-objective evaluation added validation tests based on the residuals as a second objective.

3.2.4 Fitness Functions used to Learn Dynamic Models

Several of the above papers used multi-objective approaches to assist in learning the dynamic behaviour [9][27][53][54]. In all cases, at least one objective involved the sum-of-

errors from the targeted time series. Borrelli et al. [9] considered a small set of statistical features in some of the objectives.

Ross [55] reported a lack of success in evolving a stochastic system which targeted cyclic behaviour. One reason for this was attributed to the fitness function which made use of the traditional sum-of-errors approach.

3.3 Feature-based Search Spaces

3.3.1 Context

Features have been used to define search spaces in machine learning tasks such as data mining, signal and image processing, and classification, particularly when noisy signals or considerable amounts of data are involved. Since time series data introduce dependencies between sequential values which can influence the type of features chosen to describe them, this section will focus on feature-based search spaces based on temporal information.

Time series data show up in many fields such as engineering, scientific research, finance and medicine [4]. Distance measures are required for many machine learning tasks applied to time series including model building, pattern recognition, classification and clustering. Examples of applications in which features have been used in distance measures have been grouped into the following three areas and are described in subsequent sections:

1. Feature-based fitness functions to learn dynamic models
2. Feature-based similarity measures for clustering and classification of time series
3. Feature extraction via GP for classification of time series

3.3.2 Feature-based Fitness Functions to Learn Dynamic Models

Sections 3.1 and 3.2 identified some related work which made use of statistical features in fitness functions to infer dynamic models. In all cases, features were applied when dealing with stochastic behaviour introduced through noisy target data or probabilistic models. Here is a synopsis:

- Chu [12] and Leier et al. [41] used features to evolve probabilistic GRN models exhibiting oscillating behaviour.
- Borrelli et al. [9] used a combination of statistical features as one of the objectives in a multi-objective GP which targeted time series with added noise. In their

experiments, the first objective adopted the standard sum-of-errors approach, while the second objective combined the sum-of-errors from two statistical features. For the feature-based objective, two sets of features were considered, namely mean plus standard deviation, and skew plus kurtosis.

3.3.3 Feature-based Similarity Measures for Clustering and Classification of Time Series

Tasks such as clustering, classification and search and retrieval, often related to data mining activities, make use of similarity measures [39] [43]. The following papers serve as examples illustrating how features have been used as a basis for various similarity measures:

- Wang et al. [67] made use of global, statistical features of time series to cluster several benchmark time series data-sets . Drawing from a comprehensive set of 13 features, feature subset selection was performed using a greedy forward search to identify a reduced set of features which improved clustering performance.

- Wang et al. [68] extended the above approach to cluster human motion data, depicting 10 activities, transformed into multivariate time series. Several clustering algorithms were applied and the feature vectors proved to cluster accurately and efficiently.

- Alcock & Manolopoulos [2] used the Euclidian distance from a set of features to evaluate the similarity between control chart patterns with added noise. Features included first and second order statistical features of the time series.

- Proposing that features would be less sensitive to noise if they were not based on individual time points, Nanopoulos et al. [48] used 8 first and second order statistical features as input to a neural network to classify control chart patterns.

- Extending the above research, Lavangnananda & Piyatumrong [38] added 2 more first order features aimed to better discern between noisy increasing and decreasing behaviour. As well, a further set of second order features obtained from smoothed data was added, bringing the total number of features fed into the neural network to 18. Improvement in classification accuracy was reported.

- Kadous [31] combined global features and comprehensible events to classify multivariate hand gesture signals.

- Dellaert et al. [14] explored various sets and subsets of pitch and rhythm features of speech signals to classify 4 emotions. Feature subset selection improved the classification performance.

3.3.4 Feature Extraction via GP for Classification of Time Series

Another popular use of features is found in feature extraction, where primitive features are combined to produce a similarity measure for subsequent classification purposes. GP has been used extensively to perform this task. Examples involving time series are listed below:

- Sun et al. [60] evolved classifiers to perform fault diagnosis in the fuel system of diesel engines.
- Silva & Tseng [58] evolved classifiers for different seafloor habitats based on acoustic backscatter signals.
- Lopes [45] evolved classifiers to recognize epileptic patterns in human electroencephalographic signals.

Chapter 4

Problem Statement

4.1 Statement

The focus of the research in this thesis is to explore the effectiveness of a feature-based fitness function which employs statistical features in a genetic programming application to evolve stochastic π-calculus gene gate models of gene regulatory networks. The fitness function is based on the sum-of-errors from a targeted set of statistical features characterizing the temporal gene expression levels of the simulated model. Drawing from a large set of comprehensive features, this fitness function is designed to be capable of dealing with a variety of behaviours found in gene regulatory networks such as oscillating and non-oscillating behaviours.

4.2 Justification

Much research effort is being made into modelling gene regulatory networks in order to gain an understanding of the complex interactions taking place at the cellular level. It has been recognized that stochasticity is an integral component of gene regulatory networks due to the low number of biomolecules involved [18]. One probabilistic GRN model developed recently incorporates modular gene gate components built from stochastic π-calculus expressions, a process algebra which models concurrent events [7]. Genetic programming is a machine learning technique which provides a framework to effectively evolve programs, particularly conducive to those with modular components. However, as pointed out by Ross [55], the stochastic nature of these networks poses challenges to GP. One such challenge presents itself in the fitness function, as the ability of the standard approach using sum-of-errors from the time course values is limited in the case of oscillating behaviour.

Similarity measures based on sets of statistical features of time courses have been used in clustering and classification by Wang et al. [67] and Nanopoulous et al. [48]. A sum-of-errors measure based on statistical features offers a promising approach for a fitness function dealing with stochastic behaviour. The use of features in fitness functions in this manner has been limited in GPs to date. Borrelli et al. incorporated a small number of basic features in a multi-objective GP [9] for symbolic regression with added noise, while Chu [12] and Leier et al. [41] used single features to specifically target oscillating behaviour of probabilistic models. The fitness function explored in this thesis makes use of a larger set of features and is tested on a variety of behaviours produced by both expressions with added noise and probabilistic networks.

4.3 Value

Development of a versatile, feature-based fitness function will add an alternative fitness function approach for future search and optimization problems involving stochastic and noisy systems. For the specific GP task at hand, it will provide a tool for discovery and model development. As computational power increases, there will be an ability and interest to model more complex real-life behaviour which do not behave deterministically. This research effort is a contribution in developing approaches to deal with this challenge.

Chapter 5

Feature-based Fitness Function

In the GP algorithm, each individual is assigned a fitness value which reflects how closely it behaves relative to the target. As noted in Chapter 3, a common approach for evaluating time series is based on a sum-of-errors where the error is the difference between the value of the dependent variable produced by the candidate expression and the corresponding target value. Since stochastic effects and noise can introduce phase shift and signal variation, the performance of this standard approach can be significantly degraded. As well, the random element produces a different trajectory during each simulation, making it difficult to maintain a consistent measure of fitness for the same expression.

To illustrate the degree of signal variation introduced by stochasticity, Figures 5.1 and 5.2 overlay the results of 2 simulations of the same channel and expression for two of the targets subsequently studied in this thesis. These graphs clearly demonstrate how a fitness function based on the sum-of-errors of the signal would result in a dismal score even when the target expression is encountered.

In order to overcome the deficiency of the standard fitness function approach, a fitness function based on statistical features of the signal is proposed. Using features to characterize the signal has many benefits:

- practical and easy to comprehend and implement
- robust against noisy data
- serves to lower the dimensionality of the data
- statistics allow for complex behaviours to be described with simpler numeric values backed by a large field of study
- offers a flexible approach that it is capable of handling missing data and comparing time series with different lengths

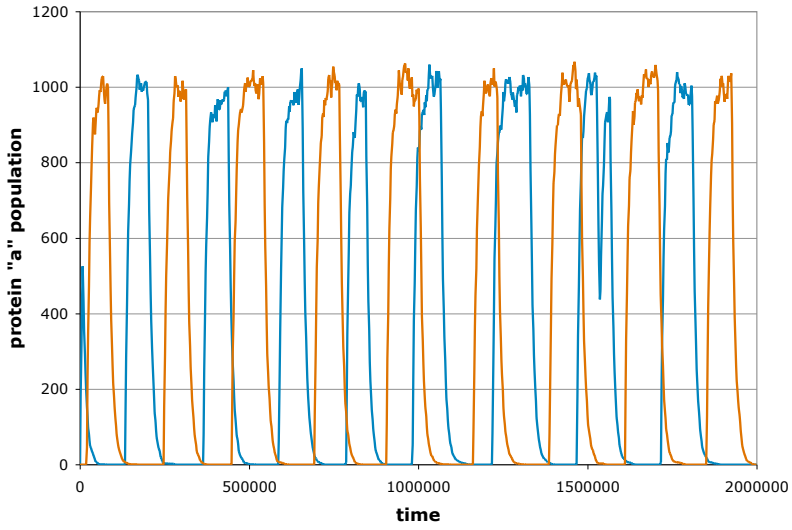


Figure 5.1: Signal from Protein “a” from Two Repressilator Simulations

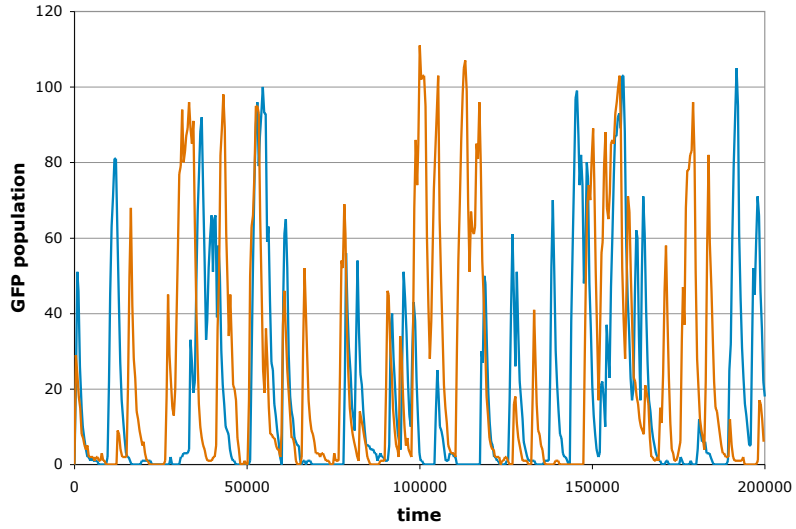


Figure 5.2: Signal from Protein “GFP” from Two D016 Simulations

- a comprehensive set of features should be able to differentiate between many varieties of time series, including oscillating and monotonic trajectories
- features can be tailored to suit the problem and prior knowledge can be incorporated

In order to develop the feature-based fitness function for each problem, the following elements were addressed:

1. Determine the features (how many and which ones) to characterize the behaviour.
2. Given the features, determine a sum-of-errors formula to evaluate the fitness.
3. Determine how many times the simulation should be repeated to obtain the overall fitness. Averaging the results of several evaluations serves to reduce the variation encountered when measuring the fitness of a single individual.

The way in which these considerations were handled are described in the remaining sections of this chapter.

5.1 Features

5.1.1 Full Set of Features

In order to determine which features to include in the fitness function, a full set of features to draw from was first defined. A set of 17 statistical features listed in Table 5.1 was adopted from previous work by Wang et al. [67] and Nanopoulos et al. [48]. Together, these features create an expressive set from which a subset tailored to suit the problem at hand can be selected. Features were calculated from the time series closely following the approach described in [67].

For this particular implementation, calculation of each feature assumed that the time series were roughly the same duration and that the values were set at evenly spaced intervals (except the mean). R [52], a popular open source system which performs statistical computation, was used as noted in Appendix C to efficiently determine some of the characteristics.

It should be noted that for many of the features, the method chosen to quantify the feature could be questioned, because often there are several ways to go about measuring the feature and some approaches involve parameters. However, a flexible aspect of this feature-based fitness function, is that it can accommodate such differences or even errors, as long as the feature is calculated consistently for all candidate expressions. Perhaps an error in the formula or departure in approach is so significant that it could be considered a different feature altogether. Perhaps the error may manifest itself in feature values such that they vary significantly from evaluation to evalutation of the same expression. This latter effect would be sorted out during the selection of the subset of features to be actually used in the fitness function, as discussed in a subsequent section.

Table 5.1: Full Set of 17 Features

1. mean	10. mean (tsa)[a]
2. standard deviation	11. standard deviation (tsa)
3. skew	12. skew (tsa)
4. kurtosis	13. kurtosis (tsa)
5. serial correlation	14. serial correlation (tsa)
6. non-linearity	15. non-linearity (tsa)
7. chaos	16. trend
8. self-similarity	17. seasonality
9. periodicity	

[a] tsa: trend and seasonally adjusted

5.1.2 Mean

The mean, μ, is the average value of the signal over the total time. In earlier experiments, with the intent to improve accuracy, the mean was calculated before the time series was evenly-spaced and based on the integral (sum of area under the curve) with the data points connected linearly, averaged by the total time:

$$\mu = \frac{0.5}{t_n - t_1} \sum_{i=2}^{n} \left[(y_i + y_{i-1}) \, (t_i - t_{i-1}) \right]$$

where n is the number of data points

(y_i, t_i) are the data points

Although this increase in accuracy did not appear to have a significant impact on the results, it was decided to continue calculating the mean in this manner.

5.1.3 Standard Deviation

Standard deviation reflects the degree to which the signal varies from the mean over the total time. Standard deviation, σ, was calculated as follows:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} (y_i - \mu)^2}{n - 1}}$$

where n is the number of data points

y_i are the evenly-spaced data points

5.1.4 Skew

Skew is a measure of how assymmetrical the data points lie around the mean. A symmetrical distribution, such as a normal distribution, has a skew of zero. If the distribution of data points is skewed to the right of the mean, characterized by a tail extending to the right, then skew is positive. Conversely, if the distribution of data points is skewed to the left of mean, then skew is negative. Skew was calculated as follows:

$$skew = \frac{1}{n\sigma^3} \sum_{i=1}^{n} (y_i - \mu)^3$$

where n is the number of data points

y_i are the evenly-spaced data points

μ is the mean

σ is the standard deviation

Skew was protected from infinite values by setting $\sigma = 0.001$ if the actual standard deviation was equal to zero.

5.1.5 Kurtosis

Kurtosis is a measure of how peaked or flat the distribution of data points is relative to the normal distribution. Kurtosis was calculated in a manner such that a normal distribution would correspond to zero, peaked data would have positive kurtosis, and flat data would be negative:

$$kurtosis = \frac{1}{n\sigma^4} \sum_{i=1}^{n} (y_i - \mu)^4 - 3$$

where n is the number of data points

y_i are the evenly-spaced data points

μ is the mean

σ is the standard deviation

Kurtosis was protected from infinite values by setting $\sigma = 0.001$ if the actual standard deviation was equal to zero.

5.1.6 Serial Correlation

Serial correlation measures whether the time series is correlated to itself over small lags and can be used to distinguish between white noise and correlation within a signal. The approach taken to quantify serial correlation was based on the Box-Pierce statistic, a portmanteau test, which takes into consideration a range of lags. The methodology and lag range (1 to 20) as described in [46] was followed. It was decided to omit multiplying the sum of the squared autocorrelations by the time series length, because this factor would increase the variability of the feature value between evaluations of the same expression. The number of data points recorded in SPiM simulations varies slightly from simulation to simulation of the same expression and there was no need to introduce this variability into the feature measure.

Serial correlation was calculated as follows:

$$serial\ correlation = \sum_{k=1}^{20} \left(r_k\right)^2$$

where r_k is the autocorrelation for lag k

$$r_k = \frac{\sum_{i=k+1}^{n} \left[\left(y_i - \mu\right)\left(y_{i-k} - \mu\right)\right]}{\sum_{i=1}^{n} \left(y_i - \mu\right)^2}$$

n is the number of data points

y_i are the evenly-spaced data points

μ is the mean

According to this formula, serial correlation is a positive value. A value equal to zero indicates noise (no correlation). Since an autocorrelation, r, ranges from 1 to -1, a serial correlation approaching 20 would indicate extremely strong correlation. To avoid infinite values, serial correlation was set to 20 for the special case of a horizontal line.

5.1.7 Non-linearity

Non-linearity is a characteristic which is examined when constructing time series models for forecasting. It can help to decide whether a linear or non-linear model is appropriate. As selected by Wang et al. [67], Teräsvirta's neural network test [62], which has a null hypothesis of linearity, was used to quantify non-linearity. Large values of this feature indicate non-linearity, while values approaching zero indicate linearity.

5.1.8 Chaos

Chaos describes behaviour which may appear random, but actually is deterministic and highly sensitive to initial values. The average Lyapunov exponent which measures the rate of divergence of nearby trajectories is used to quantify chaos. Its value is negative if behaviour converges towards stability, zero if in steady-state, and positive for divergent, chaotic behaviour.

The method described by Hilborn [26] was followed with modifications to consider multiple lags, while keeping computational time reasonable:

$$chaos = \frac{1}{n} \sum_{l=lag_1}^{lag_n} \left[\frac{1}{N} \sum_{k=k_1}^{k_N} \lambda\left(x_k, l\right) \right]$$

where n is the number of lags considered

N is the number of initial values considered

l is the lag

x is the evenly-spaced time series

$\lambda\left(x_k, l\right)$ is the Lyapunov exponent

The Lyapunov exponent is defined as:

$$\lambda\left(x_k, l\right) = \frac{1}{l} \ln \frac{d_l}{d_0}$$

where x_k is the initial point considered in the time series

$d_0 = |x_j - x_k|$

$d_l = |x_{j+l} - x_{k+l}|$

j is chosen such that $|x_j - x_k|$ is minimized, yet non-zero

for $(j - k) = 2\%$ to 20% of the time series' length

Lag, l, ranged from 5% to 20% of the time series' length, while x_k, the initial point in the time series ranged from the first point up to 0.6 of the the time series' length. For time series up to 1000 data points in length, these ranges (l and k) were traversed with an increment of 1. For longer time series, the increment was increased to $\lfloor \frac{length}{500} \rfloor$.

5.1.9 Self-similarity

Self-similarity measures long-range dependency within a time series, as quantified by the Hurst exponent [69]. The Hurst exponent was calculated as $d + 0.5$ where d is obtained by fitting the data to a fractional autoregressive integrated moving-average FARIMA$(0, d, 0)$ model. The Hurst exponent ranges between 0 and 1. Its value reflects the nature and degree of predictability found in the time series. A value of 0.5 indicates a random sequence of values, a value less than 0.5 indicates that the behaviour tends to correct itself such that it results in a certain long-term mean value, and a value greater than 0.5 signifies behaviour which is following a trend.

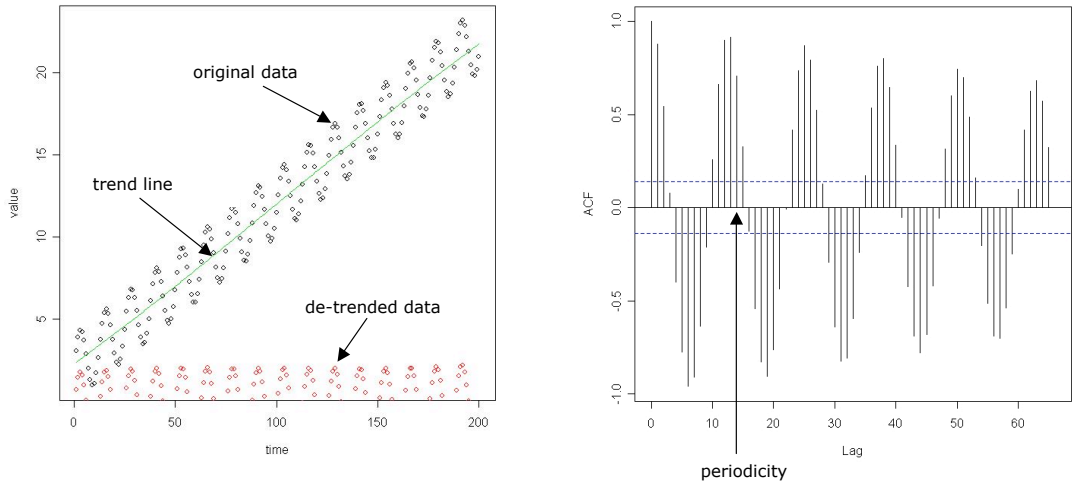
5.1.10 Periodicity

Periodicity detects cyclic patterns in the time series. This measure accommodates cyclic activity which varies in frequency, in contrast with seasonality which only considers patterns with a constant period. The algorithm presented by Wang et al. [67] was followed. The steps along with implementation details are as follows:

1. Detrend the time series by fitting a cubic smoothing spline to the time series.
2. Determine the autocorrelation function (r_k, see formula in Section 5.1.6) for lags up to $1/3$ of the time series length.
3. Look for peaks and troughs in the autocorrelation function.
4. Set periodicity to the lag which corresponds to the first peak with the following conditions met:
   (a) The peak is preceded by a trough
   (b) The difference between a trough and a peak is ≥ 0.1
   (c) The peak has positive correlation

This procedure is shown pictorially in Figure 5.3.

When no seasonal pattern is detected, the periodicity is set to 1. Otherwise, the periodicity takes a maximum value of $1/3$ the time series length. The actual time represented by one interval in the time series may vary slightly among SPiM simulations. To correct this small discrepancy, the periodicity obtained from this algorithm was subsequently multiplied by the time-step between the data points, a value which was determined when the time series was rendered evenly-spaced (through linear interpolation).



(a) De-trended Data

(b) Autocorrelation Function

Figure 5.3: Periodicity Example

5.1.11 Trend and Seasonally Adjusted Features

In the analysis of time series, a common operation is to decompose the time series into trend, seasonal and remainder components. Decomposition of the time series was performed for two purposes. Firstly, measures for trend and seasonality are derived from decomposed elements. Secondly, since a clearer picture of the data sometimes results from de-seasonalized and de-trended data, several of the features determined from the raw data were also determined from the remainder. These features are referred to as trend and seasonally adjusted (tsa). The following is a list of tsa features included in the full set of 17 characteristics:

1. mean
2. standard deviation
3. skew
4. kurtosis
5. serial correlation
6. non-linearity

In the approach taken by Wang et al. [67], the time series was first subjected to Box-Cox transformation, followed by additive decomposition. Decomposition on the transformed time series breaks the time series into trend, seasonal and remainder components:

$$Y^* = T + S + R$$

where Y^* is the transformed time series

T is the trend component

S is the seasonality component

R is the remainder component

Box-Cox transformation renders the data roughly normal and is defined as:

$$Y^* = \frac{Y^\lambda - 1}{\lambda}$$

where Y^* is the transformed time series

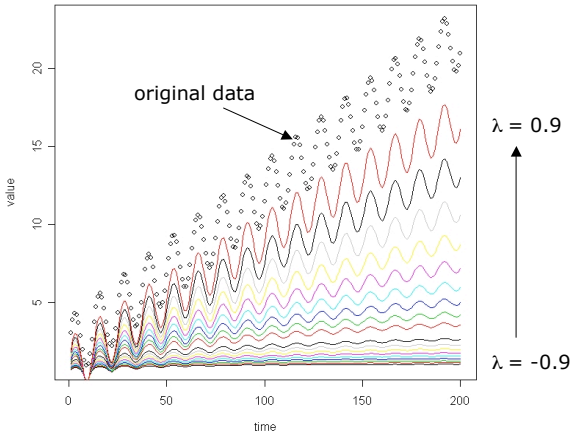
Y is the original time series. To ensure non-zero values,

values, Y_i, were set to $max(Y_i, 0.001 * Y_{max})$

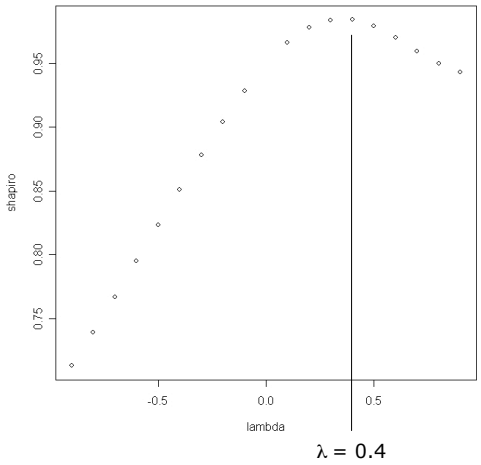
λ is the transformation parameter

The transformation parameter, λ, was non-zero and selected from the range -0.9 to $+0.9$ (considered in 0.1 increments) such that the Shapiro-Wilk statistic on the remainder, R, was maximized. The choice of this λ corresponds to the remainder element which has a distribution closest to normal.

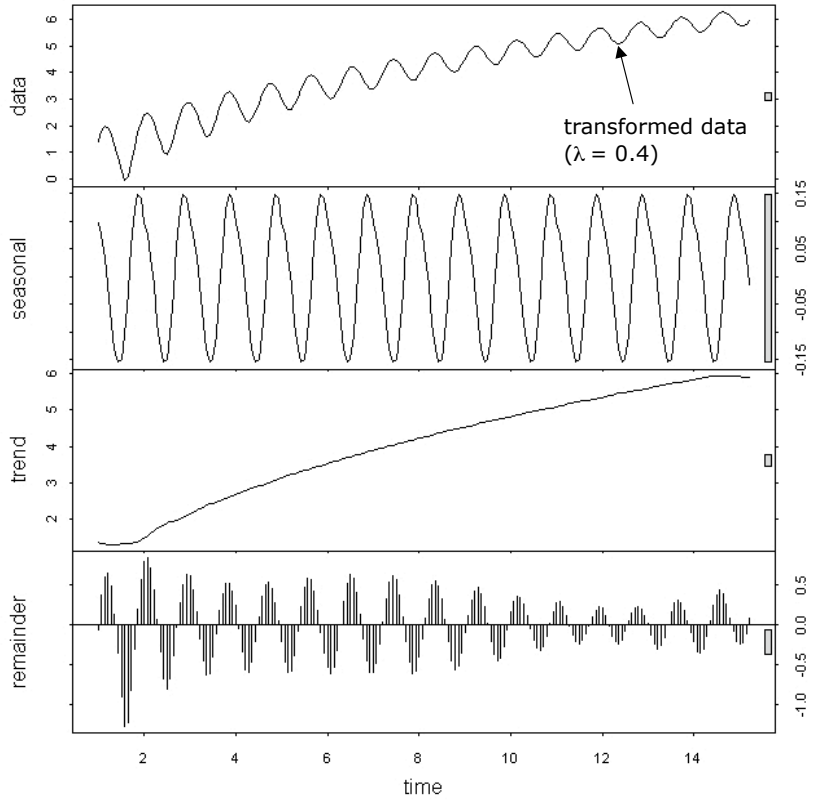
An example of the decomposition process is shown in Figure 5.4.



(a) Box-Cox Transformation



(b) Shapiro-Wilks Statistic vs. Lambda



(c) Decomposition of Transformed Data

Figure 5.4: Decomposition Example

5.1.12 Trend and Seasonality

Trend is present when there is a long-term change in the mean. Seasonality is a pattern that repeats over fixed periods of time. Given that the transformed time series, Y^* is decomposed into the trend, T, seasonal, S, and remainder, R, components such that $Y^* = T + S + R$, trend and seasonality are calculated as follows:

$$trend = 1 - \frac{\sigma^2(R)}{\sigma^2(T+R)}$$

$$seasonality = 1 - \frac{\sigma^2(R)}{\sigma^2(S+R)}$$

where σ^2 is the variance

5.1.13 Testing of Feature Calculations

After the features were coded, testing of the implementations were carried out on 27 test time series:

- 14 time series encompassing simple, benchmark behaviours (including random, linear, trending, and oscillating behaviour).
- 7 time series obtained from the internet corresponding to those tested and documented in Wang's thesis [66].
- 6 times series resulting from SPiM simulations of the gene gate expression for the repressilator, an oscillating network which is involved in subsequent experiments.

Through this testing, approaches and parameters in the feature calculations were adjusted to ensure that the feature values were being generated as expected, in a consistent manner within reasonable computational time.

5.1.14 Selecting Features from the Full Set

Tailored to suit the specific problem at hand, a subset of this full set of 17 features was selected for direct use in the fitness function. Favourable features had low coefficients of variation and were judged to be relevant to the target behaviour, as well as non-redundant. The coefficient of variation is the ratio of the standard deviation over the mean. It provides a measure reflecting the stability of the values of a feature over several evaluations of the

same expression. This coefficient was often used as a guide to select features except in the case where the mean value was close to zero.

The number of features in the subset was initially determined based on the degree of difficulty of the target expression. Simpler expressions tended to perform adequately with a smaller subset. If the chosen subset produced comparable fitness values for several other expressions as well as the target, further features were added to the subset to more definitively identify the target when it was reached.

It is anticipated that a formal means of feature subset selection would yield a more effective fitness function. Unfortunately, this is out of the scope of this thesis.

5.2 Fitness Function Formula

Given a time course of values, the fitness is determined by first calculating the subset of features for the data, and then performing a sum-of-errors on the features through the following formula:

$$fitness\ score = \sqrt{\sum_{i=1}^{n} \left(\frac{F_{i,target} - F_i}{Norm_{i,target}} \right)^2} \tag{5.1}$$

where F is the value of the feature

n is the number of features

$Norm$ is the normalization factor

Normalization was applied in order to bring the errors into a range that was common between the features, thus preventing individual features from dominating the fitness evaluation. Two approaches towards normalization were considered:

1. normalization by the average value of the target feature, $F_{i,target}$
2. normalization by standard deviation of the target feature, $\sigma_{i,target}$

Normalization by Average

Normalizing by the average is a common approach which expresses the error as a fraction of the target value. Target values close to zero can produce division-by-zero errors or large fitness contributions. When this situation was encountered, one was added to both the target and calculated features in order to avoid these detrimental effects.

Normalization by Standard Deviation

Normalizing by the standard deviation is analogous to the standard or z-score ($\frac{x-mean}{\sigma}$) [47]. It takes into account the variation inherent in the target feature itself since it expresses how far away the candidate's feature is from the target mean in terms of the target's standard deviation. When the candidate expression matches the target, the contribution of each feature to the overall fitness is equal, on average.

5.3 Number of Repeated Evaluations

Every simulation of a single expression is subject to stochastic effects, yielding a different set of feature values for each evaluation. This produces overall fitnesses which vary from simulation to simulation . A common approach to reduce variation due to noise is to repeat evaluations and average the resulting fitnesses [29]. For the experiments contained in this thesis, the fitnesses from either 1 or 4 samples were averaged to obtain the overall fitness. It was necessary to keep this number low for run-time considerations.

A side study documented in Appendix A was carried out examining the effect of sample size on GP performance. Interestingly, it found a statistically significant reduction in performance when a larger number of evaluations was taken, indicating that an increased number of samples doesn't necessarily lead to improved performance, and that some degree of noise in the fitness function may, in fact, be beneficial.

Chapter 6

Symbolic Regression Experiments

6.1 Introduction

Symbolic regression is an established genetic programming application in which mathematical expressions are evolved to produce a specified behaviour. Since adding Gaussian noise to the evaluation of expressions produces behaviour similar to that encountered in stochastic network simulations, a series of initial experiments were performed to assist in developing the feature-based fitness function, and to confirm its viability. As well, considering the GP language applied in these experiments, it could be argued that an evolved expression, $f(x)$, could also be regarded as a time series, $f(t)$.

Three symbolic regression targets were considered, encompassing both oscillating and non-oscillating behaviour. For two of the targets, which involved simple expressions, the performance of the feature-based fitness function was compared to that of the standard sum-of-errors approach.

6.2 Target Expressions

The following 3 target expressions were considered:

1. Non-oscillating: $x^4 + x^3 + x^2 + x$, through the interval $[-1, 1]$
2. Oscillating: $1 + \sin 3x$, through the interval $[0, 2\pi]$
3. Spherical Bessel function j_1: $sinx/x^2 - cosx/x$, through the interval $[0.1, 20.0]$

The first two targets were inspired from early work on symbolic regression [35]. The quartic polynomial is a commonly-used benchmark expression, while the oscillating expression was chosen because it was simple, yet had no basic equivalent expressions which

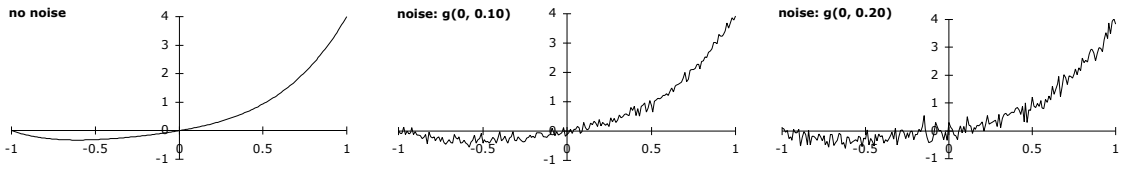


Figure 6.1: Non-oscillating Target

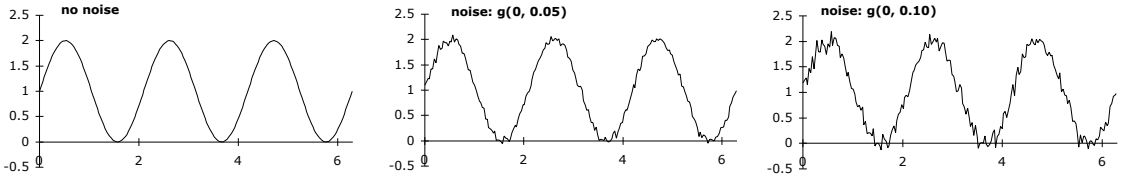


Figure 6.2: Oscillating Target

could complicate the search space. The third target, a Bessel function which oscillates with an attenuating amplitude, was tested out of curiosity to see whether features could evolve this pattern of behaviour.

The leftmost graphs in Figures 6.1, 6.2 and 6.3 plot the target expressions through their corresponding intervals.

6.3 Added Noise

At each point that a candidate expression was evaluated, Gaussian noise, $g(0, s)$, was added, where g is a Gaussian random number with zero mean and standard deviation, s. Noise levels considered in the first two experiments corresponded to standard deviation levels of roughly 2.5% and 5% of the range of target values within the interval considered, while the Bessel experiment applied noise at the 5% level only. To help visualize these quantities, Figures 6.1, 6.2 and 6.3 show sample evaluations of the target expressions with the corresponding levels of noise added.

For the simple oscillating target, $1 + sin3x$, an additional type of noise was included in

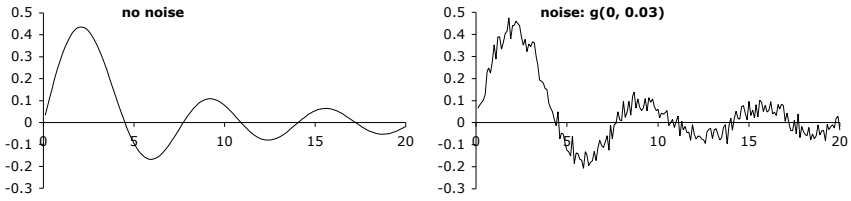


Figure 6.3: Bessel Target

Table 6.1: Noise Considered in each Experiment

Experiment	Targeted Expression	Added Noise	Added Lag
Non-oscillating	$x^4 + x^3 + x^2 + x$	none $g(0, 0.1)$ $g(0, 0.2)$	none
Oscillating	$1 + sin3x$	none $g(0, 0.05)$ $g(0, 0.10)$	none $g(0, \pi)$
Bessel Function	$sinx/x^2 - cosx/x$	$g(0, 0.03)$	none

order to simulate variations in phase which can be introduced by stochastic processes. This noise was applied by evaluating the expression over a lagged interval shifted by $g(0, \pi)$ such that the entire oscillating curve was moved horizontally.

Table 6.1 summarizes the combinations of noise considered in each experiment. It is important to point out that the noise was applied to the evaluation of each candidate expression, while the targeted expression and its corresponding targeted feature values remained deterministic since they were evaluated without any added noise.

6.4 Fitness Function

Each candidate expression was evaluated at 201 (200 for the Bessel target) evenly-spaced points over the interval. For runs with added noise, 4 evaluations per expression were carried out and the resulting fitnesses were averaged to obtain the final score. The goal of the GP was to minimize fitness, with the lowest attainable score of zero.

Feature-based Fitness Function

The feature-based fitness function as described by equation 5.1 in Section 5.2, normalized by the target feature values, was applied. The following feature subsets were selected for the targets:

1. non-oscillating: mean, standard deviation, skew
2. oscillating: mean, standard deviation, skew, kurtosis, periodicity, seasonality
3. Bessel: mean, standard deviation, skew, serial correlation, chaos, self-similarity, periodicity, trend

Table 6.2: Symbolic Regression Features used in the Fitness Function

no.	feature	target value [a] (no noise)	with noise		
			average	standard deviation	inverse coeff. of variation
Non-oscillating [b]					
1	mean	0.533	0.534	0.014	38.2
2	standard deviation	1.104	1.122	0.014	79.7
3	skew	1.484	1.416	0.040	35.7
Oscillating [c]					
1	mean	1.000	1.000	0.007	145.1
2	standard deviation	0.707	0.714	0.007	100.9
3	skew	0.000	0.003	0.023	0.15
4	kurtosis	-1.507	-1.451	0.022	-67.4
5	periodicity	2.073	2.083	0.016	132.8
6	seasonality	0.996	0.985	0.003	306.5
Bessel function [d]					
1	mean	0.048	0.048	0.002	21.8
2	standard deviation	0.151	0.154	0.002	77.5
3	skew	1.164	1.096	0.048	22.8
4	serial correlation	10.061	9.301	0.124	74.8
5	chaos	0.108	0.112	0.005	21.0
6	self-similarity	0.999	0.998	0.000	8793.2
7	periodicity	6.600	6.453	0.141	45.8
8	trend	0.816	0.794	0.045	17.8

[a] increased precision was used in GP runs
[b] values with noise: based on 500 runs with $g(0, 0.2)$ added noise
[c] values with noise: based on 200 runs with $g(0, 0.10)$ added noise
[d] values with noise: based on 500 runs with $g(0, 0.03)$ added noise

Values for the target features were obtained by evaluating the expression without noise at 201 (200 for the Bessel target) evenly-spaced points over the interval.

These subsets were chosen by examining the features from multiple evaluations of the target function with noise added. Features with high inverse coefficients of variation were favoured, as this indicated an amount of stability amid the noise. As well, since the target feature values were based on those without noise, features whose average values were not significantly affected by the addition of noise were also taken into consideration. Table 6.2 lists the selected features along with their target values and corresponding average, standard deviation and inverse coefficient of variation values when noise was applied.

Table 6.3: Symbolic Regression Function and Terminal Sets

target	non-oscillating	oscillating	Bessel
function set	$+, -, *, \%,$ sin, cos, exp, ln	$+, -, *, \%, sin$	$+, -, *, \%,$ sin, cos, exp, ln
terminal set	x	$x, 1$	x

Standard GP Fitness Function

A sum-of-absolute-errors approach was used for the standard GP fitness function:

$$fitness\ score = \sum_{i=1}^{n} |f_{target}(x_i) - f(x_i)|$$

where $f_{target}(x)$ is the target expression, $f(x)$ is the expression being evaluated (including the added noise, if present) and n is the number of evenly-spaced points over the interval. For the non-oscillating and oscillating targets, n was set to 201, while for the Bessel target n was set to 200 for all evaluations.

6.5 GP Function and Terminal Sets

Functions and terminals for the simple targets were identical to those used in similar problems (simple symbolic regression and the trigonometric identity problem) in [35]. The Bessel target applied the same sets as the non-oscillating target in order to examine how a change in target features impacts what the GP evolves. Table 6.3 lists the function and terminal sets for each target. $\%$ and $\ln$ were protected functions to ensure closure.

6.6 GP Parameters and Settings

For all experiments, a common set of GP parameters was applied as listed in Table 6.4, with the exception of an increased population and maximum number of generations for the Bessel target, considering that it was a more complex expression and behaviour to target.

Table 6.4: Symbolic Regression GP Parameters

population	500 (non-oscillating and oscillating targets) 1000 (Bessel target)
maximum no. of generations	20 (non-oscillating and oscillating targets) 30 (Bessel target)
probability of crossover	0.9
probability of mutation	0.1
probability of reproduction	0.0
elitism	none
selection	tournament (size 3)
initial population	ramped half and half
min. initial tree depth	2
max. initial tree depth	6
maximum tree depth	17
prob. crossover point is branch	0.9
max. regenerative depth for mutation	5
max. number of retries	50
no. evaluations fitness score averaged over	4

6.7 GP Software

GP runs were performed on Open BEAGLE software [23], a C++, object-oriented, generic framework for performing Evolutionary Computation. It supports tree-based, strongly-typed genetic programming. Aside from defining the function and terminal sets, most of the supplementary code required to customize the system for each problem involved the fitness function. Further implementation details are provided in Appendix C.

6.7.1 Typical Run-times

Run times were highly dependent on the computer system, the features in the fitness function, the number of times the expression was evaluated per fitness score, the GP parameters (population, maximum generations), and the expressions encountered during the GP run. For the non-oscillating target with the feature-based fitness function and noise added, runs typically took less than 5 minutes. For the oscillating target, with the feature-based fitness function and noise added, runs generally completed under 2 hours. Runs for the Bessel target took just over 4 hours.

Table 6.5: Non-oscillating Target Results

	Fitness Function	Added Noise	No. Runs Target Found (of 20)	95% Confidence Interval for the Run Success Rate	Average Generation Target Found
1	feature	none	5	11% - 47%	10.8
2	feature	$g(0, 0.1)$	6	14% - 52%	10.3
3	feature	$g(0, 0.2)$	6	14% - 52%	10.7
4	standard	none	9	26% - 66%	13.0
5	standard	$g(0, 0.1)$	6	14% - 52%	12.7
6	standard	$g(0, 0.2)$	5	11% - 47%	13.4
baseline	feature	none	0	0% - 19%	—

6.8 Results

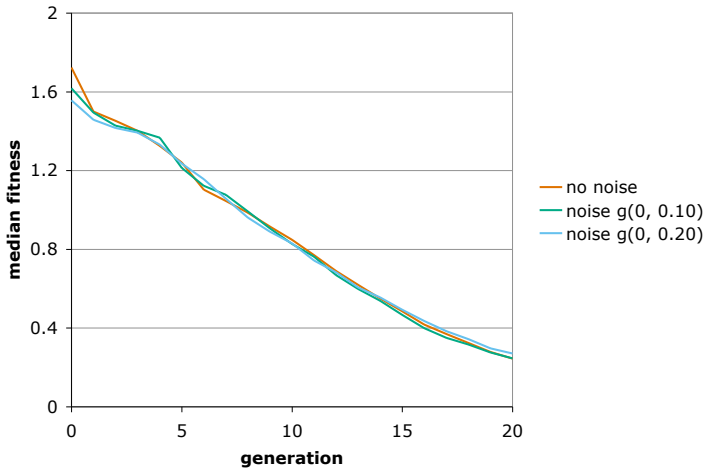
In addition to the 20 runs completed for the oscillating and non-oscillating targets, baseline runs with tournament size 1 were also carried out using the feature-based fitness function to confirm that the targets could not be constructed as frequently through random selection alone. Baseline results are included in the tables. As well, for each configuration, plus four confidence intervals for the run success rates were calculated (Appendix D) and presented in the tables.

6.8.1 Non-oscillating Target

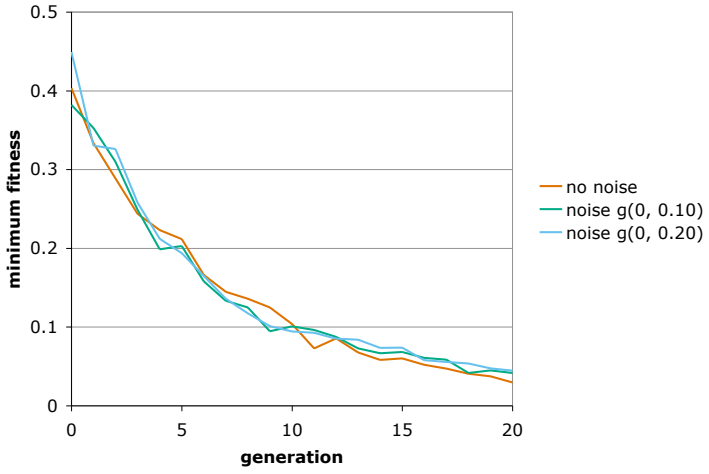
Twenty runs per configuration were executed and the results are shown in Table 6.5. The median and best-of-generation fitnesses averaged over all runs are illustrated in Figure 6.4 for the feature-based fitness function and Figure 6.5 for the standard GP fitness function. For the standard approach, as the noise levels increased, the GP converged to higher (worse) fitness values. This was due to the noise.

Interestingly, two of the feature-based GP runs yielded the mirror image expression, $x^4 - x^3 + x^2 - x$, which received near-zero scores since it exhibited the same characteristics as the target. If these mirror expressions had been considered acceptable, then the feature-based fitness function tally would have increased to [6, 6, 7] successful runs corresponding to the [0, $g(0, 0.1)$, $g(0, 0.2)$] levels of added noise, respectively.

Without added noise, the standard GP fitness function was the superior performer for symbolic regression of the non-oscillating target. As noise was added to the candidate expressions, both fitness functions appeared to be performing similarly.

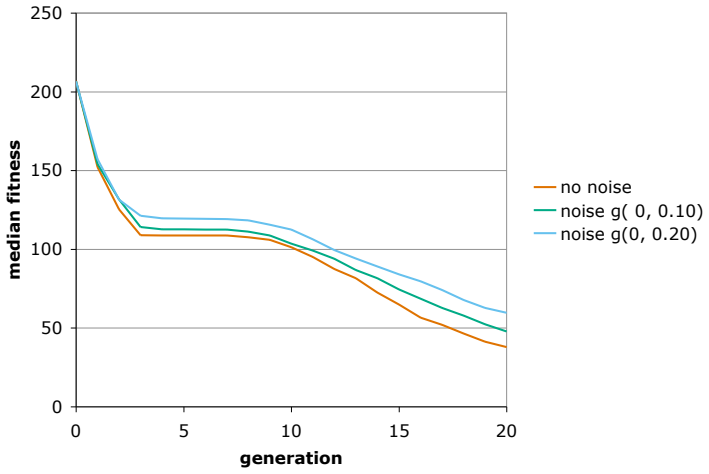


(a) Average Median-of-Population Fitness by Generation

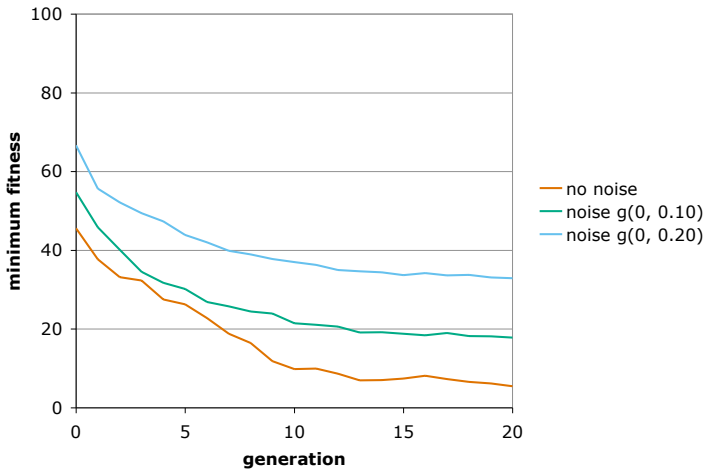


(b) Average Best-of-Population Fitness by Generation

Figure 6.4: Non-oscillating Target with the Feature-based Fitness Function: GP Results Averaged over 20 Runs



(a) Average Median-of-Population Fitness by Generation



(b) Average Best-of-Population Fitness by Generation

Figure 6.5: Non-oscillating Target with the Standard Fitness Function: GP Results Averaged over 20 Runs

Table 6.6: Oscillating Target Results

	Fitness Function	Added Noise	Added Lag	No. Runs Target Found (of 10)	95% Confidence Interval for the Run Success Rate	Average Generation Target Found
1	feature	none	none	10	67% - 100%	9.5
2	feature	$g(0, 0.05)$	none	9	57% - 100%	12.3
3	feature	$g(0, 0.10)$	none	8	48% - 95%	8.3
4	feature	none	$g(0, \pi)$	9	57% - 100%	9.7
5	feature	$g(0, 0.05)$	$g(0, \pi)$	10	67% - 100%	6.8
6	feature	$g(0, 0.10)$	$g(0, \pi)$	9	57% - 100%	9.9
7	standard	none	none	2	5% - 52%	5.5
8	standard	$g(0, 0.05)$	none	0	0% - 33%	—
9	standard	$g(0, 0.10)$	none	0	0% - 33%	—
baseline	feature	none	none	1	0% - 43%	6

6.8.2 Oscillating Target

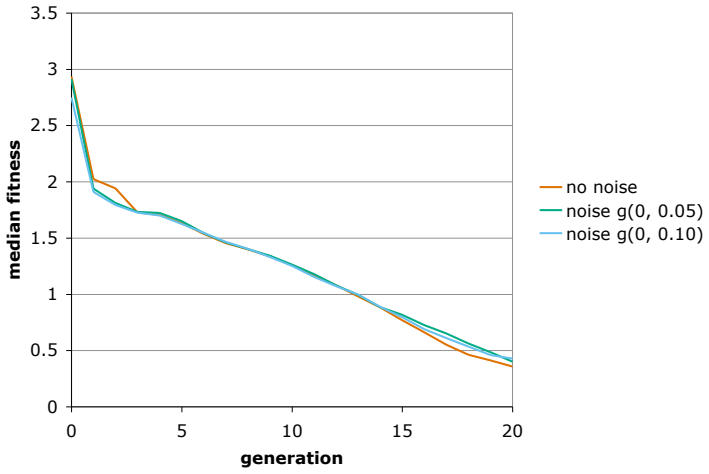
Ten runs per configuration were executed for the oscillating target. It was considered that the target was found if any function in the form $1 + sin(c \pm 3x)$ (where c is any constant) was constructed. Expressions of this form exhibit the same characteristics. Results are listed in Table 6.6 and the median and best-of-generation fitnesses averaged over all runs are plotted in Figure 6.6 and Figure 6.7 for the feature-based fitness function, without and with added lag respectively, and Figure 6.8 for the standard GP fitness function.

In contrast with the standard fitness function's poor performance, the feature-based fitness function was found to be extremely successful for all noise levels, regardless of the varying amounts of significant lag introduced at each evaluation.

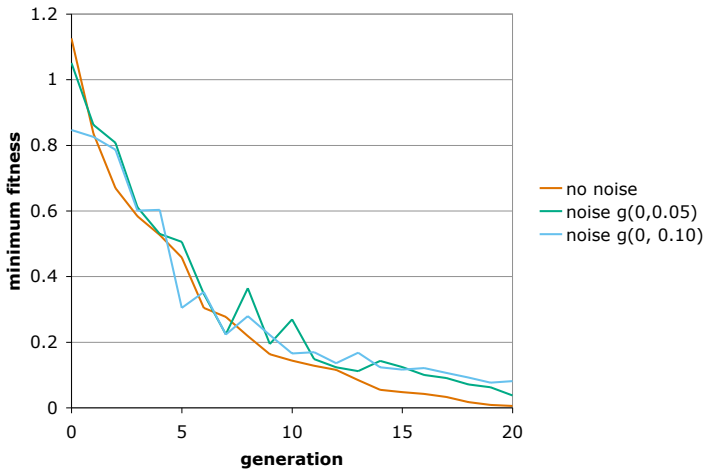
6.8.3 Bessel Function Target

Twenty runs were performed at a noise level of $g(0, 0.03)$ which corresponds to approximately 5% of the range of target values over the interval evaluated. The median and best fitnesses of the population by generation averaged over all the runs are shown in Figure 6.10. Although the GP was not successful at evolving the target expression, $sinx/x^2 - cosx/x$, expressions with favourable fitnesses exhibited attenuating and oscillating behaviour. Figure 6.9 displays two such examples, which correspond to the following relatively straightforward expressions:

A: $\frac{1}{x} sin\left[xsin\left(\frac{x(e^x - x)}{x(e^x + x) + sin(sinx) - lnx}\right)\right]$

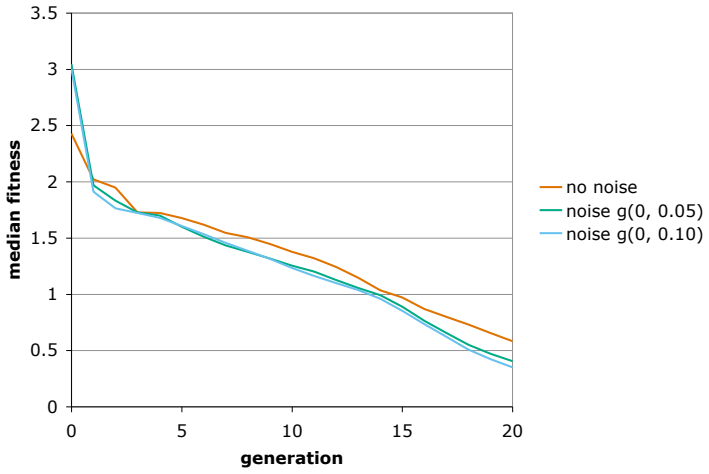


(a) Average Median-of-Population Fitness by Generation

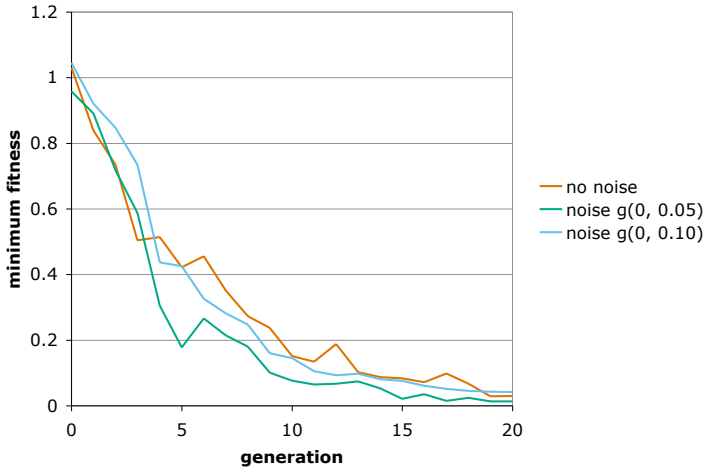


(b) Average Best-of-Population Fitness by Generation

Figure 6.6: Oscillating Target with the Feature-based Fitness Function (no Lag): GP Results Averaged over 20 Runs

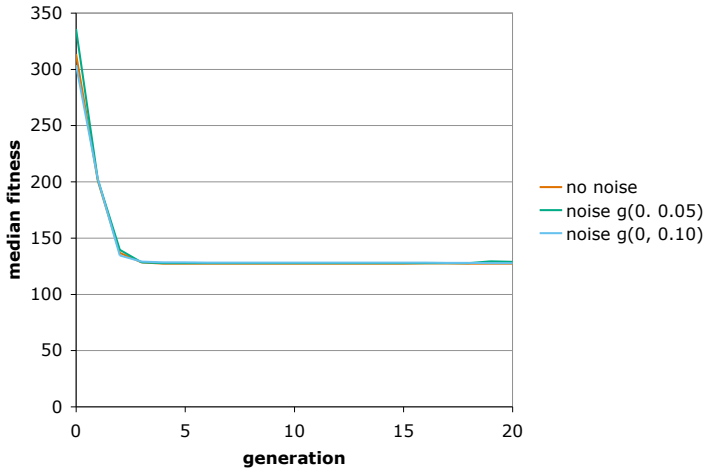


(a) Average Median-of-Population Fitness by Generation

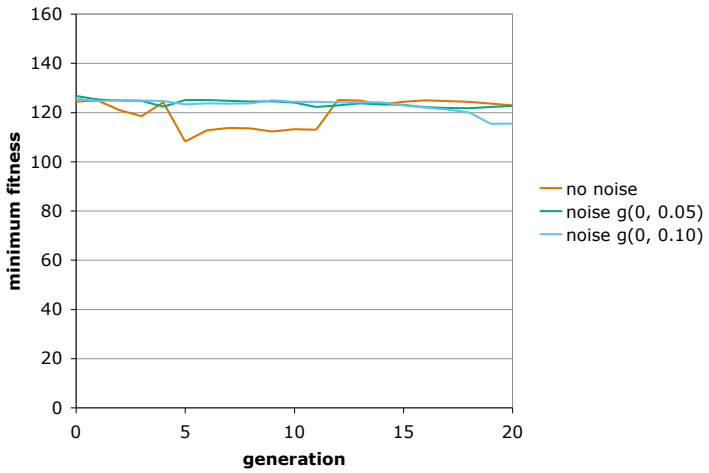


(b) Average Best-of-Population Fitness by Generation

Figure 6.7: Oscillating Target with the Feature-based Fitness Function (with Lag): GP Results Averaged over 20 Runs



(a) Average Median-of-Population Fitness by Generation



(b) Average Best-of-Population Fitness by Generation

Figure 6.8: Oscillating Target with the Standard Fitness Function (no Lag): GP Results Averaged over 20 Runs

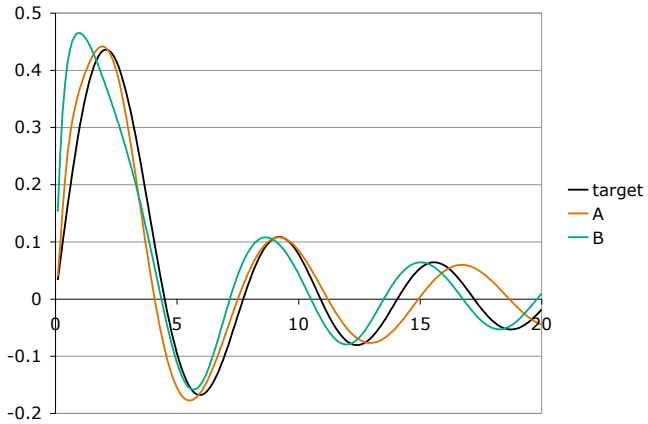
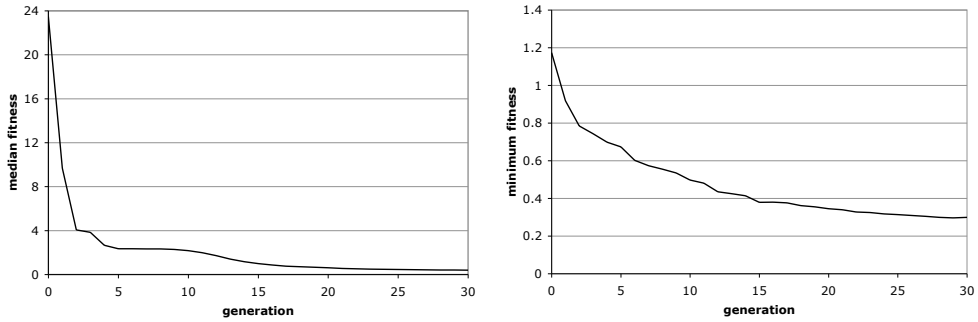


Figure 6.9: Two Evolved Expressions Compared to the Bessel Target

B: $sin\left[\frac{sin\left(x+\frac{sinx}{x}-1\right)}{x+cos1}\right]$



(a) Average Median-of-Population Fitness by Generation

(b) Average Best-of-Population Fitness by Generation

Figure 6.10: Bessel Target: GP Results Averaged over 20 Runs

6.9 Discussion

In these experiments, the target features were obtained by evaluating the target expression without any added noise. This decision was made so that an apples-to-apples comparison to the standard fitness function approach could be carried out. Targeting noiseless features is a departure from the experiments involving stochastic networks which are presented in

the next chapter. Since the stochasticity is inherent in the simulation of the model, rather than realized through added noise, the stochastic network experiments had no choice but to target feature values which included the effects of stochastic behaviour.

For the symbolic regression experiments, it can be argued that the noiseless feature approach served to increase the difficulty of the search. Added noise affects some of the feature values, such as serial correlation, chaos and self-similarity, to such a degree that the noiseless target features may no longer reflect the behaviour of the expression with noise. If this is the case, it is expected that the feature-based fitness function's performance will degrade as the level of noise is increased unless some compensation for the noise is added to the target feature values or if only features are chosen whose average values are not affected by the noise.

For the non-oscillating target, the 3 features were sufficient to distinctly describe the target expression when there was no added noise. Except for the mirror image, no other expression obtained near-zero scores. However when noise was added, a few other expressions managed to obtain low scores comparable to the target. Perhaps the addition of further features to the fitness function would help to more distinctly distinguish the target expressions from others, and lead to an increased number of hits. The fitness function for the very successful set of runs targeting the oscillating expression made use of 6 features.

When noise was added, the final fitness score was set to the average of 4 expression evaluations to combat the variation encountered. Upon review of the results, there was no indication that this choice was detrimental, so this value was carried through to subsequent experiments.

The standard sum-of-errors approach to symbolic regression is suitable for noiseless data [35], and in fact may be more efficient than the use of features. However, as shown in these experiments, when considering noise, performance of the standard evaluation is compromised. Similar results have been shown in [9] [13].

Results of the experiments demonstrate the ability of the feature-based fitness function to perform symbolic regression in the presence of noise. Particular strength in this fitness function was observed for the oscillating target.

Although the target expression was not evolved for the Bessel target, the feature-based fitness function was able to evolve more complex behaviour by simply changing the subset of targeted feature values. Solutions with the best scores were evolved in latter generations. As a consequence, despite the shrink mutation, they were typically longer expressions containing a significant amount of bloat (redundant code). Perhaps a parsimony term added to the fitness function would help to explore the search space of shorter expressions more thoroughly.

Based on the positive results from these experiments, the next step was to apply the feature-based fitness function to the synthesis of stochastic networks. These experiments are described in the next chapter.

6.10 Further Work

It would be interesting to investigate how much noise could be tolerated by the GP for the oscillating and non-oscillating targets when using the feature-based fitness function. For the Bessel function target, experimenting with the addition of a parsimony term to the fitness function may help the GP to successfully synthesize the exact target expression.

In general, because of its relatively fast run-times, symbolic regression with added noise could serve as a test-bed for further studies into several aspects of the feature-based fitness function such as determining a beneficial subset of features and the number of repeated evaluations to average the fitness over.

6.11 Supporting Documentation

The following supporting materials are provided in the Appendices and on the accompanying DVD:

- full set of features for each target expression
- openBeagle files tailored for each target expression
- output and log files for all symbolic regression runs (including openBeagle reports, if generated)
- fitness function and GP implementation details
- confidence interval calculations

Chapter 7

Gene Gate Experiments

7.1 Introduction

The next set of experiments involved using GP and the feature-based fitness function to infer GRN behaviour using the stochastic gene gate model described in Section 2.3. Genetic programming is a well suited machine learning technique to learn this particular model because the tree structure of its individuals can readily piece together modular components and produce expressions of variable length.

Varying degrees of model abstraction were applied to the following three targeted gene gate networks which were which were selected from [7]:

1. Repressilator: An oscillating network in which the rates as well as the targeted expression are evolved.

2. D016: A non-oscillating network exhibiting irregular, spiky behaviour.

3. D038: A non-oscillating network with noisy behaviour.

These networks offer a variety of stochastic behaviours upon which to study the effectiveness and versatility of the feature-based fitness function. In this chapter, separate sections are devoted to detailing the set-up and outcome of each experiment.

7.2 Repressilator

7.2.1 Target Network

The repressilator is a well-known synthetic oscillating network which behaves like a biological clock. It was described in [17] and stochastically simulated using gene gates in

[7] and [8]. The repressilator gene network is illustrated in Figure 7.1 (with notation per Figure 2.2) and is described by the following gene gate expression:

$$\text{neg(a, b)} \mid \text{neg(b, c)} \mid \text{neg(c,a)}$$

Its behaviour is characterized by alternating cycles of expression of three proteins present in a system of three gates arranged such that a cascading effect is created when the product of one gate represses the production of a protein in another (see Section 2.3.3 for further description).

A parameter analysis in [8] identified constraints on the rates which lead to regular and unperturbed oscillation. Rates for the target network were selected in accordance with these constraints and set at $\epsilon = 0.1$, $\delta = 0.0001$, $r = 100.0$, and $\eta = 0.001$. Behaviour of the repressilator network simulated with this set of rates is illustrated in Figure 7.2.

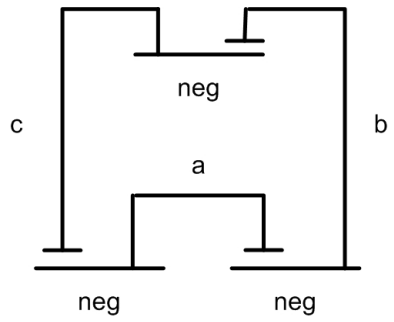


Figure 7.1: Repressilator Gene Gate Network

7.2.2 Fitness Function

Fitness was based on a single channel of information from the SPiM simulation of the candidate expression. Preferential order of the channel used in the fitness function was first “a”, “b”, then “c”. The duration of each simulation was 2,000,000 time units over which approximately 1000 data points were recorded. Prior to determining the features from a simulation, the first 5% of the data was ignored to remove initial effects. During the start-up of a SPiM simulation, there is a “warm-up” period which doesn’t reflect the steady-state behaviour of the model. Except for the calculation of the mean, the time series was rendered evenly spaced over the remaining points through linear interpolation.

Features from a set of 200 simulations of the target expression were examined and 5 features were selected for use in the evaluation. The features along with their mean, standard deviation and inverse coefficient of variation are listed in Table 7.2.2.

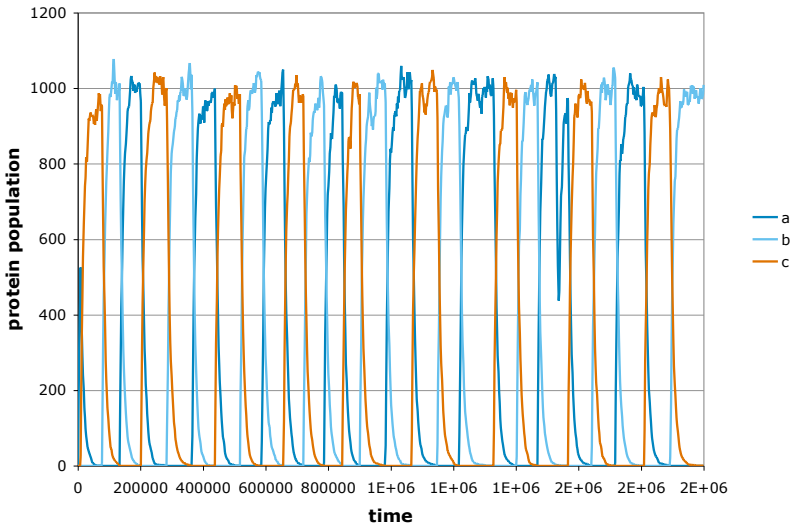


Figure 7.2: Typical Repressilator SPiM Simulation

Table 7.1: Repressilator Features used in the Fitness Function [a]

no.	feature	average	standard deviation	inverse coeff. of variation
1	mean	333.27	20.48	16.3
2	standard deviation	422.09	8.02	52.6
3	serial correlation	10.04	0.36	27.6
4	chaos	0.052	0.005	11.4
5	periodicity	227905	8554	26.6

[a] increased precision was used in GP runs

For this experiment, feature differences in the fitness function were normalized by the target feature's standard deviation. Target values for the features were calculated from the 200 simulations. The fitness score for a candidate expression was set to the average fitness of 4 SPiM simulations.

7.2.3 GP Function and Terminal Sets

For the repressilator target, the gene gate expression as well as the rates were evolved. Since the network behaviour depends on the value of the rates relative to each other, one of the rates, ϵ was fixed at 0.1 (as was done in [8]), and the remaining 3 rates (δ, r, η) were included in the GP tree. The strongly-typed GP function set is described in Table 7.2.

Table 7.2: Repressilator GP Functions

function	type	number of parameters	parameter type	corresponding gene gate
root	$root$	2	($gate$,$rates$)	root of the tree with 2 branches: expression, rates
|	$gate$	2	($gate$, $gate$)	parallel operator
neg	$gate$	2	(ch, ch)	neg gate (see Section 2.3.2)
rates	$rates$	3	($ephemeral\ UInt$, $ephemeral\ UInt$, $ephemeral\ UInt$)	rate holder for (δ, r, η)

The root of the tree was of type $root$ whose 2 branches consisted of a gene gate expression branch and a rate branch.

There was only one terminal type for the gene gate expression, channel with type ch, which represented a channel and could take on the value of a, b, or c.

Rates were represented by unsigned integer ephemeral constants randomly selected from the interval [0, 10,000]. The actual rate was determined by a mod operation to obtain the rate exponent (with base 10). Within the GP, the rates were restricted to the following ranges:

1. δ: $10^{-3}, 10^{-4}$
2. r: $10^0 - 10^5$ inclusive
3. η: $10^{-6} - 10^0$ inclusive

These ranges were selected in order to avoid excessive simulation run-times and were based on values considered in [8].

7.2.4 Target Expression

Corresponding to this function and terminal set, the target expression was:

neg(a,b) | neg(b, c) | neg(c,a) with rates (-4, 2 and -3).

The tree, consisting of an expression branch on the left side and a rate branch on the right, is illustrated in Figure 7.3. Based on this function and terminal set, and a minimum tree depth of 3, the probability of randomly constructing the target tree is $\frac{1}{81{,}648}$.

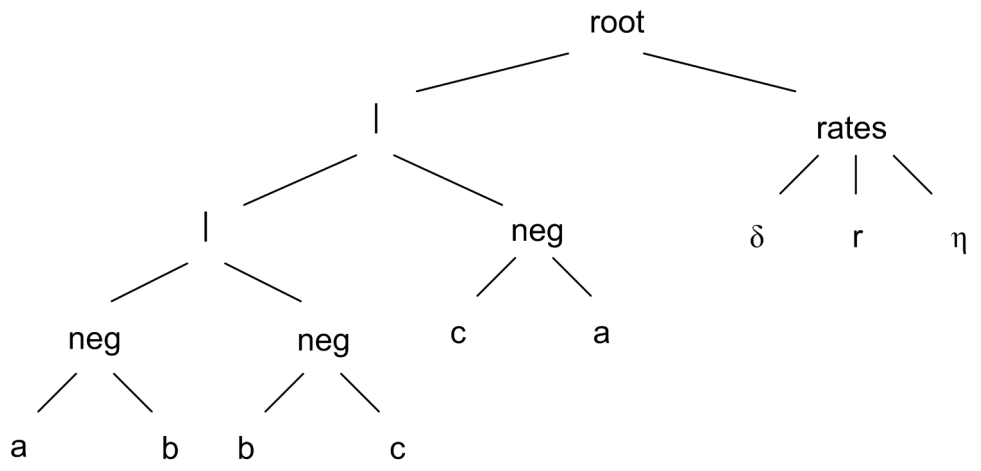


Figure 7.3: Repressilator Target GP Tree

7.2.5 GP Parameters and Settings

Table 7.3 lists parameters used for the repressilator GP runs. A tree with a depth equal to the minimum initial depth of 3 would contain a single neg gate. Population initialization used the grow method because the Open BEAGLE software couldn't construct full trees for depths greater than the fixed depth of the rate branch of the tree.

7.2.6 GP Software

Similar to the symbolic regression experiments, GP runs were performed on Open BEAGLE software [23]. Supplementary code was required to define the functions, types, and fitness function in order to customize the system for the gene gate networks. Further implementation details are found in Appendix C.

Typical Run-times

Run times were highly dependent on the computer system, the features in the fitness function, the number of times the expression was evaluated per fitness score, the GP parameters (population, maximum generations), and the expressions encountered during the GP run. For the repressilator target run-times ranged between 2 to 5 days. Most of the run-time was attributed to the SPiM simulations. Due to these lengthy run-times, the number of runs performed per configuration was limited to 20.

Table 7.3: Repressilator GP Parameters

population	200
maximum no. of generations	30
probability of crossover	0.9
probability of standard mutation	0.05
probability of shrink mutation	0.1
probability of reproduction	0.0
elitism	none
selection	tournament (size 3)
initial population	grow
min. initial tree depth	3
max. initial tree depth	6
maximum tree depth	9
prob. crossover point is branch	0.6
max. regenerative depth for mutation	6
max. number of retries	50
duration of SPiM simulation	2,000,000
no. points collected per simulation	1000
no. simulations fitness score averaged over	4

7.2.7 Results

The GP was run 20 times with the parameters set per Table 7.3. As well, 20 baseline runs were performed in which the tournament size was set to one, such that trees were subject to mutation and crossover without selection. Median population fitness and best-of-generation fitness by generation averaged over all the GP runs are shown in Figure 7.4.

As previously mentioned, Blossey et al [8] identified a set of rate constraints which result in regular and unperturbed oscillation. Preliminary analyses of the repressilator found it unable to differentiate between the behaviour of several networks whose rates met these constraints. Consequently, it could be misleading to judge the success of the GP solely based on whether the exact target was found or not.

To help assess the results, it was decided to first identify a set of rate combinations whose corresponding fitnesses could not be discerned from those of the exact target. A series of simulations were performed for several rate combinations which were receiving favourable scores, and the statistical properties of their fitnesses were determined and compared to that of the target. In all, 9 rate combinations were deemed "indiscernible" from the target as determined by two sample t-tests with confidence levels exceeding 95% (Appendix D). A visual summary of this analysis is found in Figure 7.5. In this diagram, the average fitness for several rate combinations is depicted by a circle, whose diameter is pro-

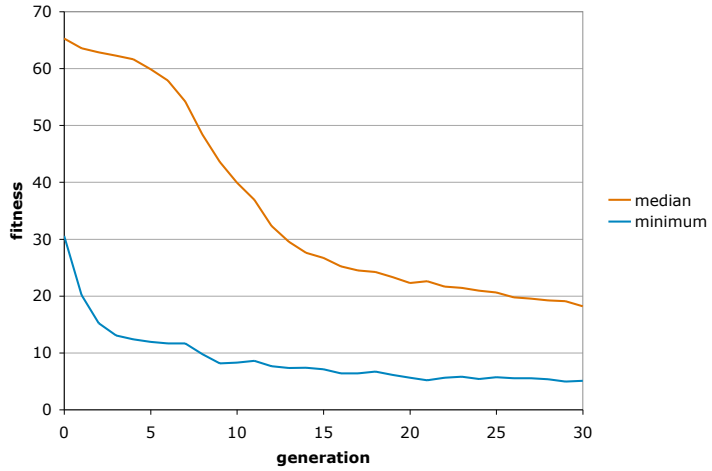


Figure 7.4: Repressilator GP Results Averaged over 20 Runs

portional to the average fitness obtained from 200 simulations. In addition to the fitness values, the behaviour of the 9 rate combinations could also not be visually distinguished from that of the target rate combination upon examination of their simulation plots.

A comparison of the results between the GP and baseline runs are presented in Table 7.4. In this table, the result from each run is grouped into one of the following 4 categories according to their degree of success:

1. Target expression and correct set of rates were found.
2. Target expression was found, but set of rates were off-target. With this rate combination, fitnesses are indiscernible from the target.
3. Target expression was found, but set of rates were off-target. Fitness was favourable ($<= 7$), but with this rate combination, fitnesses are discernible from that of the target.
4. Target expression was not found or fitness >7.

Note that the average fitness of the target, obtained from 200 simulations, was 2.2 with a standard deviation of 0.8. As such, any expression with fitness greater than 7 would not stand out when examining top individuals during any GP run.

7.2.8 Discussion

The results in Table 7.4 do not convincingly demonstrate the abilities of the feature-based fitness function. The repressilator network was used widely in preliminary studies, where

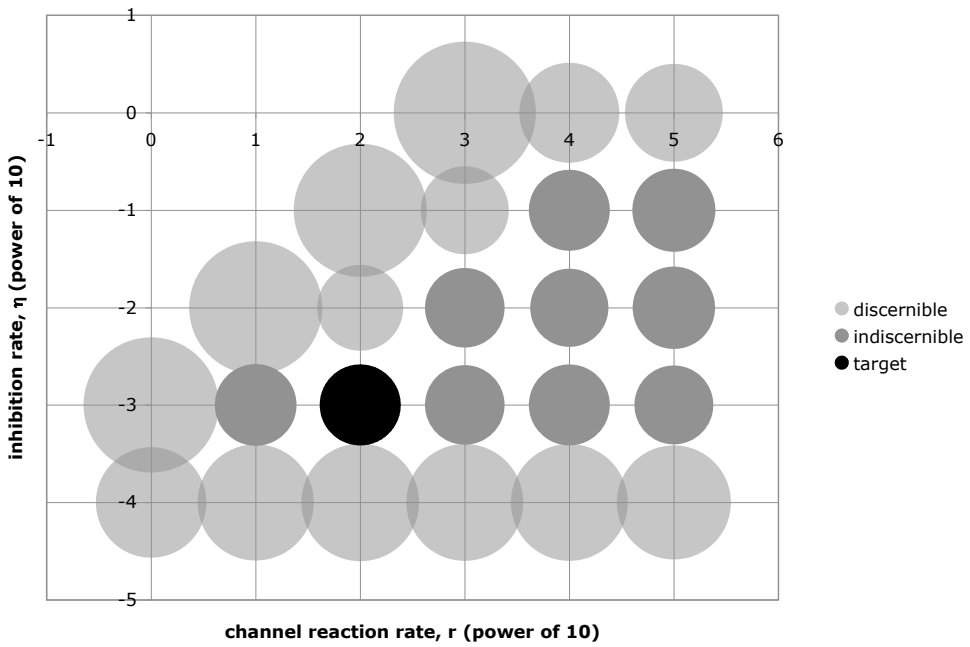


Figure 7.5: Average Fitness of Repressilator with Various Rate Combinations

it was concluded that evolving the expression alone, while keeping the rates fixed, was too simple of a problem. Although adding the rates to be evolved increased the difficulty of the problem, it does not appear to have provided sufficient challenge for the GP, since it was difficult to discern between the behaviour of several rate combinations and the target. The combined probability of randomly constructing a solution that lies in either category 1 or 2 is $\frac{1}{8{,}165}$, which is a marked increase from the probability for the exact target, $\frac{1}{81{,}648}$. This increase in probability rendered the problem too simple considering that a single GP run evaluates 6,000 individuals in total. Not much more effort beyond one run would be required to exhaustively enumerate all trees (or even randomly generate trees) until the target expression was discovered.

As well, the choice of small population size to compensate for the system's simplicity may have hindered the performance. A population of 200 is low for a GP, and may not have offered a large enough pool for the GP operators to perform effectively.

Despite this, positive observations can be made. The fitness function proved capable of assigning favourable fitnesses to the target network. Only a limited number of expressions composed of 3 gates, other than the target expression, obtained fitnesses less than 7. Furthermore, the average generation that category 1 and 2 trees were found in the full-fledged GP runs was 8.1, compared to 15.0 for the baseline runs. Although these values are based on 7 and 6 runs respectively, it suggests that selection, powered by the feature-based fitness function, caused the target to be synthesized at a faster rate.

Table 7.4: Repressilator GP Results

cat.	description	GP			baseline		
		No. Runs (of 20)	95% Confid. Interval for Run Success Rate	Avg. Gen.	No. Runs (of 20)	95% Confid. Interval for Run Success Rate	Avg. Gen.
1	target expression and rates found	2	2%-32%	5.0	2	2%-32%	13.5
2	expression found; fitness indiscernible from target	5	11%-47%	9.4	3	5%-37%	16.0
3	expression found; fitness discernible from target	8	22%-61%	12.0	6	14%-52%	9.2
4	target expression not found	5	11%-47%	–	9	26%-66%	–

7.2.9 Further Work

The target repressilator network could be made more challenging by evolving the actual rate values rather than the exponents. However, a preferable suggestion would be to find a more difficult oscillating gene gate network to target. Such a network would be in a better position to explore the capabilities of the feature-based fitness function in evolving oscillating behaviour.

7.2.10 Supporting Documentation

The following supporting documentation is made available in the Appendices or on the accompanying DVD:

- features for 1 channel over 200 simulations
- probability tree for random generation of the target expression
- statistics which determined which rate combinations were producing similar fitnesses
- openBeagle files tailored for the repressilator target
- output and log files for all repressilator runs (including openBeagle reports, if generated)
- implementation details for the GP and fitness function
- confidence interval calculations

7.3 D016

7.3.1 Target Network

D016 is a synthetic network which was experimentally identified in [25] to behave like a logical NOR gate (in the lac$^-$ strain), with inputs defined by two probes referred to as inducers and with output indicated by a fluorescent signal produced by a protein product called GFP. This network was stochastically simulated using gene gates in [7], generating the same logical behaviour exhibited in the experimental results. The D016 network is illustrated in Figure 7.6 and is described by the following gene gate expression:

negp[TetR, rtr[TetR,aTc]] | negp(LacI, rtr[LacI, IPTG]]
| negp[LacI, tr[lcI]] | negp[lcI, tr[GFP]] | rep [aTc] | rep[IPTG]

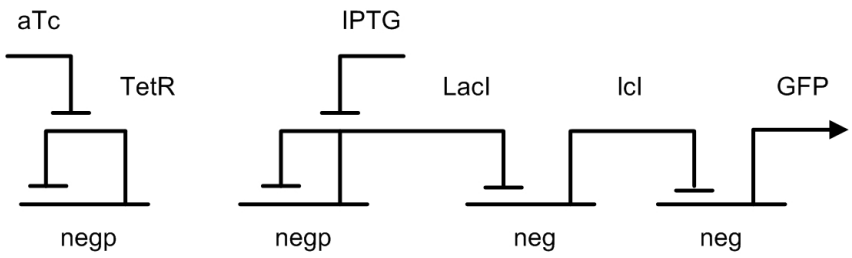


Figure 7.6: D016 Gene Gate Network

Note that the "rep" gates are the probing inducers, and rates have been omitted as parameters since they are fixed at $\epsilon = 0.1$, $\delta = 0.001$, $r = 1.0$, $\eta = 0.01$ for TetR, LacI, LambcI, GFP (the channels), and $r = 100.0$ for aTc and IPTG (the inducers).

What is curious about this circuit is how the addition of aTc affects GFP production even though the the gate it represses is seemingly not connected to the elements producing the GFP. The biological explanation for this behaviour has been discussed, but not determined [7]. The stochastic gene gate model offers a useful tool for further investigation.

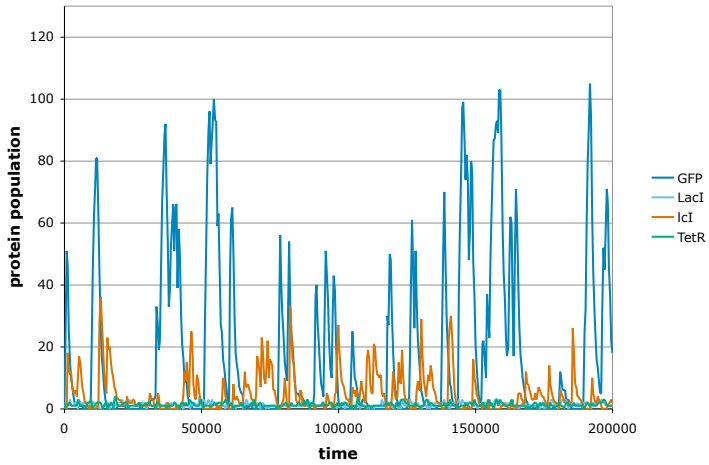
This circuit was designed to be probed by two repressing proteins, aTc and IPTG, which when applied in a Boolean manner (with either the absence or presence of each inducer) leads to 4 combinations of inputs. Since there are 4 proteins (channels) in the network and the application of each of the possible 4 probe combinations changes the behaviour of the network, a rich set of information is available to base the fitness function on.

Initial experiments found that the fitness function with features taken from two combinations could more definitively differentiate between the target and near-target expressions, compared to the single combination without inducers. Consequently, the following 2 of the

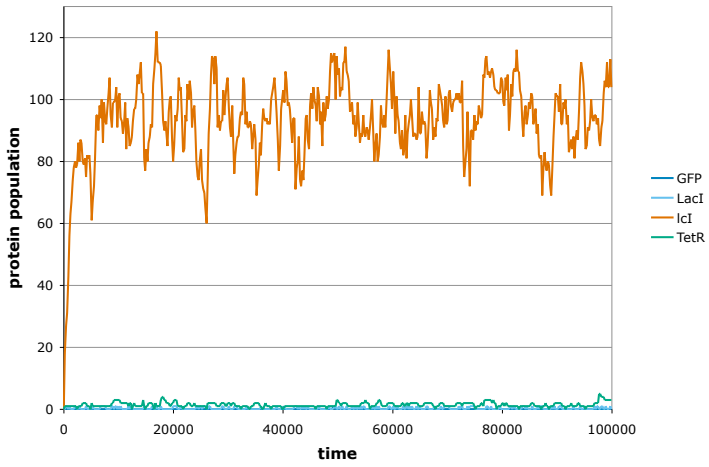
possible 4 combinations of probes were incorporated into the fitness function for this experiment:

1. without inducers
2. with inducer rep[IPTG]

Sample simulations of these combinations are found in Figure 7.7.



(a) Without inducers



(b) With inducer rep[IPTG]

Figure 7.7: Typical D016 SPiM Simulation

7.3.2 Fitness Function

To determine one fitness value for a candidate expression, 2 SPiM simulations were required, one for each probe combination incorporated in the fitness function. From these simulations, 8 time courses of gene expression levels, one from each channel, were available upon which to obtain the features. The SPiM simulation parameters were set as follows:

1. without inducers: 500 data points spread over 200,000 time units
2. with the IPTG inducer: 500 data points spread over 100,000 time units

Prior to determining the features from a simulation, the first 5% of the data was ignored to remove initial effects and, except for the calculation of the mean, the time series was evenly spaced using linear interpolation over the remaining points.

The average (μ), standard deviation (σ) and inverse coefficient of variation ($\frac{\mu}{\sigma}$) of the features resulting from a series of 200 simulations of the target expression were examined and 17 features were selected for use in the fitness function. These features are listed in Table 7.3.2. Average target feature values used in the fitness function were taken from these simulations as well.

For this particular experiment, two different normalization approaches were considered: normalization by the target feature's average and normalization by the target feature's standard deviation. As well, the number of sets of simulations over which the fitness was averaged was varied. Two situations were tested: 1 and 4 simulations. In all, 4 sets of GP runs were carried out.

7.3.3 GP Function and Terminal Sets

For this target, patterns in the $negp$ gates allowed for simplification, such that the number of parameters could be reduced. It was decided to build the transcription factors into the gate, along with the inducer protein and the rates. This lead to $negp$ gates with parameters limited to channels only. The resulting strongly-typed GP function set is described in Table 7.6. The root of each expression was of type $gate$. There was a single terminal type, ch, which represented a channel and could take on the values of a, b, c, or d, representing GFP, LacI, lcI, and TetR respectively.

Table 7.5: D016 Features used in the Fitness Function [a]

no.	channel	feature	average	standard deviation	inverse coeff. of variation
without inducers					
1	GFP	mean	20.66	4.29	4.8
2	GFP	standard deviation	27.43	2.97	9.2
3	GFP	skew	1.40	0.32	4.4
4	GFP	serial correlation	2.67	0.51	5.2
5	GFP	chaos	0.081	0.006	14.3
6	GFP	self-similarity	0.999	0.000	8419.6
7	LacI	mean	1.35	0.05	26.1
8	lcI	mean	4.00	0.61	6.6
9	lcI	standard deviation	5.36	0.80	6.7
10	lcI	chaos	0.074	0.005	14.0
11	TetR	mean	1.34	0.06	23.9
with IPTG inducer					
12	GFP	mean	0.012	0.016	0.8
13	LacI	standard deviation	0.411	0.019	22.1
14	lcI	mean	91.80	1.75	52.4
15	lcI	standard deviation	12.11	1.08	11.2
16	lcI	chaos	0.079	0.003	31.3
17	TetR	mean	1.35	0.07	19.7

[a] increased precision was used in GP runs

7.3.4 Target Expression

Corresponding to this function and terminal set, the target expression was:

$$\text{negpe(d)} \mid \text{negpf(b)} \mid \text{neg(b,c)} \mid \text{neg(c,a)}$$

The tree is illustrated in Figure 7.8. The probability of randomly generating this tree from the specified function and terminal sets with a minimum tree depth limit of 3 imposed, is $\frac{1}{139,810}$.

7.3.5 GP Parameters and Settings

Table 7.7 lists the parameters used for the D016 GP runs. A minimum initial tree depth of 3 means that all initial trees are composed of at least 2 gates in parallel.

Table 7.6: D016 GP Functions

function	type	number of parameters	parameter type	corresponding gene gate
|	gate	2	($gate$,$gate$)	parallel operator
neg	gate	2	(ch, ch)	neg(a,b) [a]
negpe	gate	1	(ch)	negpe(a) = negp(a, rates, rtr(a,aTc)) [a]
negpf	gate	1	(ch)	negpf(a) = negp(a, rates, rtr(a,IPTG)) [a]

[a] for details on neg and negp gates see Section 2.3.2

Table 7.7: D016 GP Parameters

population	500
maximum no. of generations	30
probability of crossover	0.9
probability of standard mutation	0.05
probability of shrink mutation	0.1
probability of reproduction	0.0
elitism	none
selection	tournament (size 3)
initial population	grow
min. initial tree depth	3
max. initial tree depth	5
maximum tree depth	8
prob. crossover point is branch	0.75
max. regenerative depth for mutation	6
max. number of retries	50
duration of SPiM simulation	200,000 (without inducers) 100,000 (with IPTG inducer)
no. points collected per simulation	500
no. simulations fitness score averaged over	1, 4

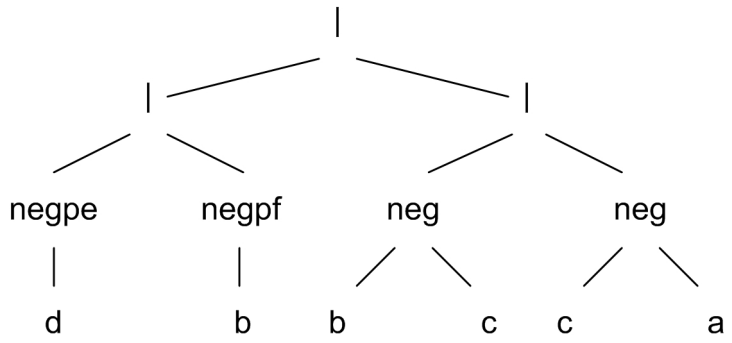


Figure 7.8: D016 Target GP Tree

7.3.6 GP Software

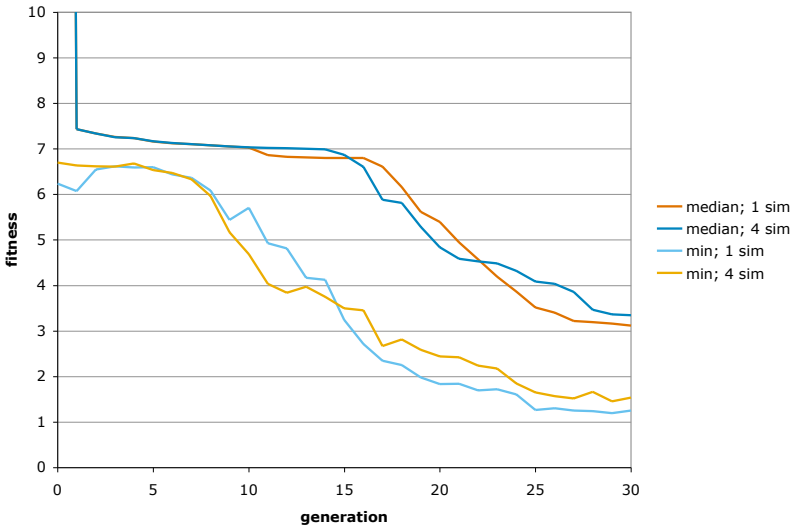
The software used to perform the GP runs was described previously in Section 7.2.6. The only departure from that section is that the tree for the D016 target contained only the network expression and not the rates.

Typical Run-times

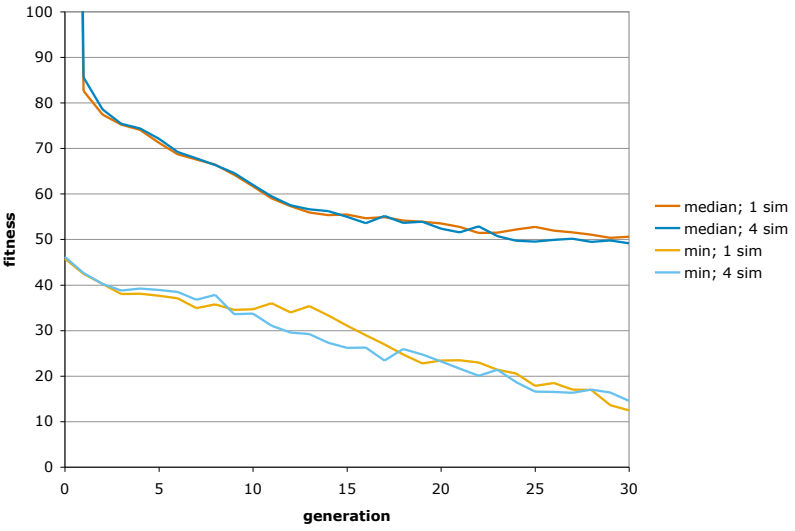
Run times were highly dependent on the computer system, the features in the fitness function, the number of times the expression was evaluated per fitness score, the GP parameters (population, maximum generations), and the expressions encountered during the GP run. For the D016 target with the fitness averaged over 4 simulations, run-times ranged between 1 and 5 days. Most of the run-time was attributed to the SPiM simulations.

7.3.7 Results

Twenty runs for each of the 4 sets were performed and results are listed in Table 7.8. Median population fitness and best-of-generation fitness by generation averaged over all runs are shown in Figure 7.9. Twenty baseline runs with tournament size 1 (as opposed to 3) were also completed. Among the 20 baseline runs, the target expression was constructed only once. In order to assess the approach to normalization and the effect of the number of simulations performed for a fitness score, the fitness of the target expression (when found) and the best-of-run fitness when the target wasn't found are summarized in Table 7.9.



(a) Fitness normalized by average



(b) Fitness normalized by standard deviation

Figure 7.9: D016 GP Results Averaged over 20 Runs

Table 7.8: D016 GP Results

	Fitness Function Normalization	Number of Simulations Averaged for Fitness	No. Runs Target Found (of 20)	95% Confid. Interval for the Run Success Rate	Average Generation Target Found
1	average	4	15	53%-89%	19.5
2	average	1	14	48%-86%	22.1
3	std. dev.	4	12	39%-78%	18.9
4	std. dev.	1	15	53%-89%	21.7
baseline	—	—	1	0%-26%	17

7.3.8 Discussion

The GP algorithm was successful in repetitively evolving the target expression with the feature-based fitness function. Results from the baseline runs and the random tree probability analysis confirmed that the target expression was not too simple for the population size, such that it could have been constructed randomly without the help of the GP operators. With a population of 500 and 30 generations in each run, 20 GP runs would generate 300,000 individuals. Finding 1 target among 20 baseline runs is judged to be consistent with the probability analysis which determined a 1 in 140,000 (approximately) chance of randomly generating the target tree in the initial population.

This experiment considered two approaches to normalization (average versus standard deviation) and also varied the number of simulations averaged to obtain the fitness score (one versus four). Because the four combinations each registered successes in such close proportion, it would require far more runs than the 20 performed here to identify any significant statistical difference in the results via a proportional comparison test. However, a comparison between the fitness scores for the target (when it was found), and the fitness scores for the best-of-run individuals when the target wasn't found, as compiled in Table 7.9, is worth examining.

Although the fitness scores are not standardized between the two methods of normalization, the numbers show that there is less overlap between fitness scores of target and near-target expressions when normalizing by the standard deviation compared to the average. This indicates that normalizing by the standard deviation is able to more definitively distinguish between the target and non-target expressions. Similarly, the same observation can be made when comparing the number of simulations averaged to obtain the fitness score for an expression. The scores resulting from 4 simulations have less overlap compared to those obtained from a single simulation, indicating improved differentiation with

Table 7.9: D016 Fitness Score Comparison

Fitness Function Normalization	Number of Simulations Averaged for Fitness	Target (when found)			Best-of-run (when target not found)		
		min.	max.	median	min.	max.	median
average	4	0.64	1.40	0.89	1.05	6.88	1.10
average	1	0.17	2.8	1.09	0.60	6.23	0.77
std. deviation	4	2.75	5.06	4.00	3.15	39.55	22.00
std. deviation	1	2.95	6.38	3.88	10.64	39.52	21.84

an increased number of simulations. These trends are also confirmed when examining the list of top-scoring individuals gathered throughout the GP runs. When there is less overlap, the target expressions tend to appear at the top of the list and consequently stand out among other expressions which are also receiving favourable scores.

Examination of near target expressions can provide insight into how well the fitness function is identifying the target behaviour. Two near target expressions were identified from the D016 runs which involved normalization by standard deviation and averaging fitnesses over 4 simulations:

1. Near target #1: neg(d,d) | negpf(b) | neg(b,c) | neg(c,a)
   This expression contains 3 of the 4 gates present in the target (neg(d,d) replaced negpe(d)) and received a score of 3.15 which falls within the range of scores obtained for the target expression, 2.75 to 5.06 (Table 7.9).

2. Near target #2: negpe(d) | negpf(b) | neg(b,c) | neg(c,a) | negpf(d)
   The expression contains all 4 gates in the target, along with one additional gate, negpf(d). It received a fitness score of 7.54 which falls outside the range of observed D016 scores.

Simulations of these expressions are found in Figure 7.10 alongside those for the target expression for comparison purposes. There appears to be no substantial difference between the behaviour of the 3 expressions. Closer study of the values of the features which contributed to the scores did not reveal any obvious differences between the target and near target #1. However, a significant difference was identified between the target and near target #2 for the mean value of the TetR protein in the simulation run without inducers. In fact, it looked like the increase in fitness experienced by this near target expression can be solely attributed to this difference. A graph focussing on this protein (Figure 7.11) confirmed the numbers.

These observations help to confirm that the feature-based fitness function is capable of identifying the target behaviour with a sensitivity that can differentiate between small changes in behaviour.

7.3.9 Further Work

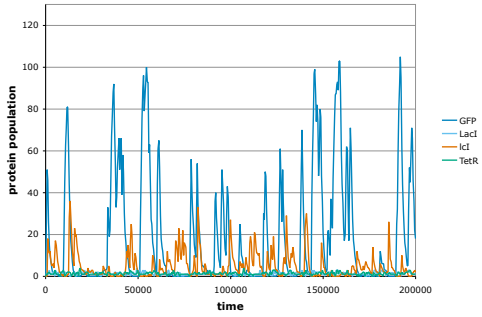
Considering that the population was at the lower limits recommended for the GP algorithm [50] and that the target expression was found quite frequently, this system could be run with increased difficulty in order to explore the limits of the feature-based fitness function. As a first step, the GP language could be made more sophisticated by increasing the parameters in the neg gates such that the rates and transcription factors are evolved. For example, in the next experiment which deals with the D038 network, the transcription factors are broken out as modular parameters within the neg gate itself.

In addition, it would be interesting to explore the effect of reducing the number of features included in the fitness function, to see whether improvements in evaluation efficiency could realized without compromising the GP's success.

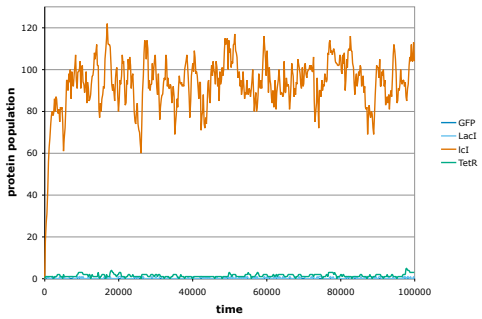
7.3.10 Supporting Documentation

The following supporting documentation is made available in the Appendices or on the accompanying DVD:

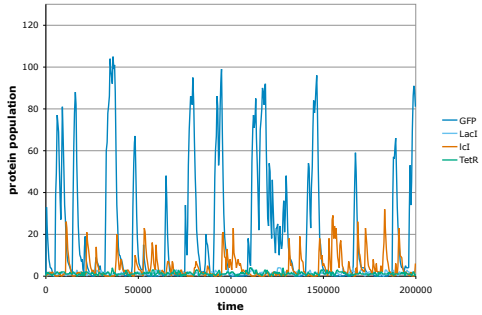
- features for 8 channels over 200 simulations
- SPiM input file for the target expression
- probability tree for random generation of the target expression
- openBeagle files tailored for D016
- output and log files for all D016 runs (including openBeagle reports, if generated)
- implementation details for the GP and fitness function
- confidence interval calculations



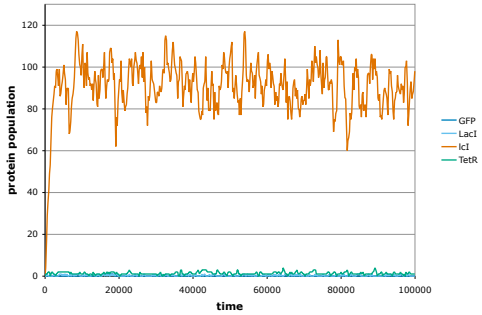
(a) Target without inducers



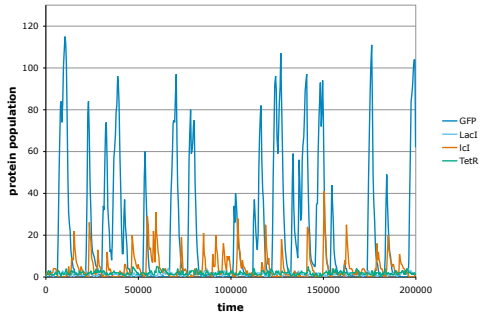
(b) Target with IPTG inducer



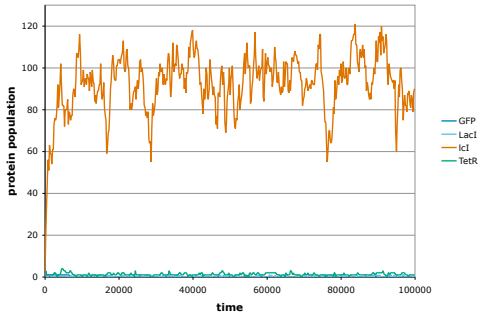
(c) Near target #1 without inducers



(d) Near target #1 with IPTG inducer

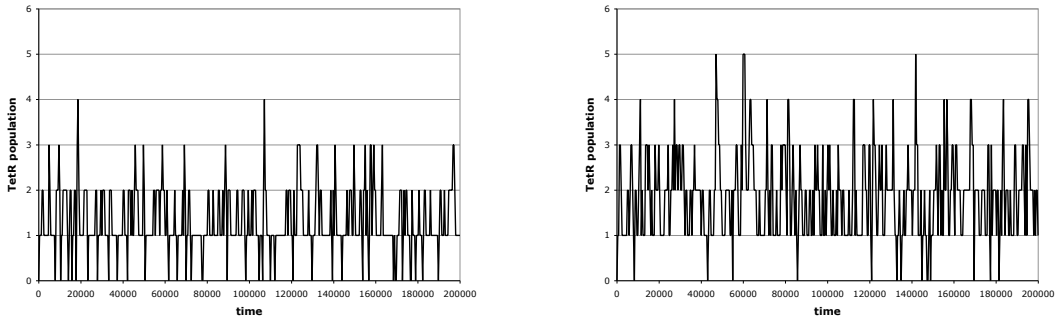


(e) Near target #2 without inducers



(f) Near target #2 with IPTG inducer

Figure 7.10: SPiM Simulations of the D016 Target and Near Target Expressions



(a) Target

(b) Near-target #2

Figure 7.11: TetR Levels from SPiM Simulations of the D016 Target and Near Target #2 without Inducers

7.4 D038

7.4.1 Target Network

D038 is another synthetic network which was described in [25] and stochastically simulated using gene gates in [7]. Similar to D016, this circuit was designed to be probed by two repressing proteins, aTc and IPTG. In the lac$^-$ strain, it behaves like a logical NOT IF gate marked by appreciable GFP production only when the inducer combination is [with aTc, without IPTG]. The D038 network is illustrated in Figure 7.12 and is described by the following gene gate expression:

negp[TetR, η_1, rtr[TetR,aTc]] | negp(TetR, η_1, rtr[LacI, IPTG]]
| negp[LacI, η_2, tr[lcI]] | negp[lcI, η_2, tr[GFP]] | rep [aTc] | rep[IPTG]
where $\eta_1 = 0.25$ and $\eta_2 = 1.0$

Note that the "rep" gates are the probing inducers, and most rates have been omitted as parameters since they remain constant for this system at $\epsilon = 0.1$, $\delta = 0.001$. $r = 1.0$ for TetR, LacI, lcI, GFP (the channels), and $r = 100.0$ for aTc and IPTG (the inducers).

Preliminary experiments found that the single combination of inducers [with aTc, without IPTG], which produces significant levels of GFP, contained sufficient information for the fitness function to identify the target expression. Behaviour of this network with this combination of inducers is shown in the sample simulation found in Figure 7.13. Although not always the case with a stochastic model, a Boolean description can be used to explain the network behaviour [25]: When aTc is present, TetR levels are low, enabling production of LacI. This in turn inhibits expression of lcI and allows production of GFP.

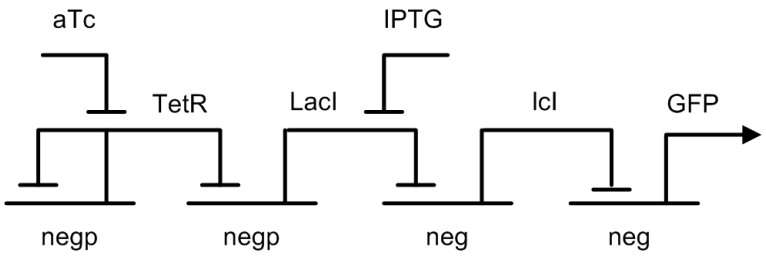


Figure 7.12: D038 Gene Gate Network

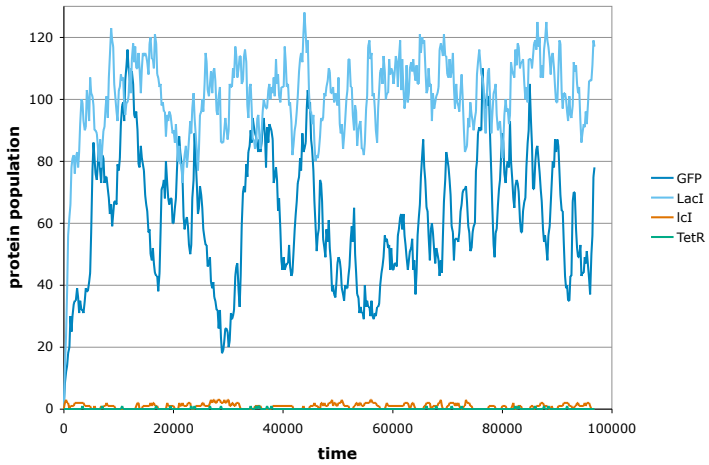


Figure 7.13: Typical D038 SPiM Simulation (with aTc & without IPTG inducers)

7.4.2 Fitness Function

Each candidate expression was simulated in SPiM for 100,000 time units in which approximately 500 data points were recorded for each of the 4 channels. Prior to determining the features from a simulation, the first 5% of the data was removed to eliminate initial effects and, except for the calculation of the mean, the time series was evenly spaced over the remaining points using linear interpolation.

Features from a set of 200 simulations of the target expression were examined and 10 features were selected for use in the evaluation. These features, along with their statistical information, are listed in Table 7.4.2. The average and standard deviations found in this table were used for the target values required in the fitness function.

For this experiment, feature differences in the fitness function were normalized by the target feature's standard deviation. The fitness score for a candidate expression was set as the average fitness taken over 4 SPiM simulations.

Table 7.10: D038 Features used in the Fitness Function [a]

no.	channel	feature	average	standard deviation	inverse coeff. of variation
1	GFP	mean	62.90	3.99	15.8
2	GFP	standard deviation	21.07	1.75	12.0
3	GFP	chaos	0.086	0.006	14.6
4	GFP	self-similarity	0.999	0.000	13230
5	LacI	mean	99.76	1.43	69.7
6	LacI	standard deviation	9.88	0.79	12.6
7	LacI	chaos	0.091	0.007	13.2
8	LacI	self-similarity	0.997	0.001	1932
9	lcI	standard deviation	0.988	0.097	10.2
10	TetR	standard deviation	0.211	0.020	10.4

[a] increased precision was used in GP runs

7.4.3 GP Function and Terminal Sets

A more modular and detailed GP language was applied to this target compared to the D016 experiment. Here, the $negp$ gate is used in its general form, and the transcription factors become parameterized with proteins and nested. The strongly-typed GP function set is described in Table 7.11. The root was of type $gate$. A new type, tf, representing a transcription factor was introduced in this function set. As well, an inducer terminal type was added, resulting in two terminal types:

1. Channel, ch, could take on the value of a, b, c, or d, corresponding to GFP, LacI, lcI, and TetR respectively

2. Inducer, ind, could take on the value of e or f, corresponding to aTc and IPTG respectively

For every candidate expression throughout the GP run, the rates were fixed per Table 7.12.

7.4.4 Target Expression

Corresponding to this function and terminal set, the target expression was:

negp(d, rtr(d,e)) | negp(d, rtr(b,f)) | negp(b,tr(c)) | negp(c,tr(a))

The tree is illustrated in Figure 7.14. The probability of randomly generating this tree with the specified function and terminal sets and a minimum tree depth of 4, is $\frac{1}{2{,}236{,}962}$.

Table 7.11: D038 GP Functions

function	type	number of parameters	parameter type	corresponding gene gate
|	$gate$	2	($gate$,$gate$)	parallel operator
negp	$gate$	2	(ch, tf)	negp gate [a]
rtr	tf	2	(ch, ind)	repressible transcription factor, rtr(b,e) [a]
tr	tf	1	(ch)	transcription factor, tr(b) [a]

[a] for details on rtr and tr network elements and negp gates see Section 2.3.2

Table 7.12: D038 Fixed Rates

rate	value
production rate, ϵ	0.1
inhibition rate, η	0.25 for the rtr transcription factor 1.0 for the tr transcription factor
degradation rate, δ	0.001
channel reaction rate, r	1.0 for TetR, LacI, lcI and GFP 100.0 for aTc and IPTG

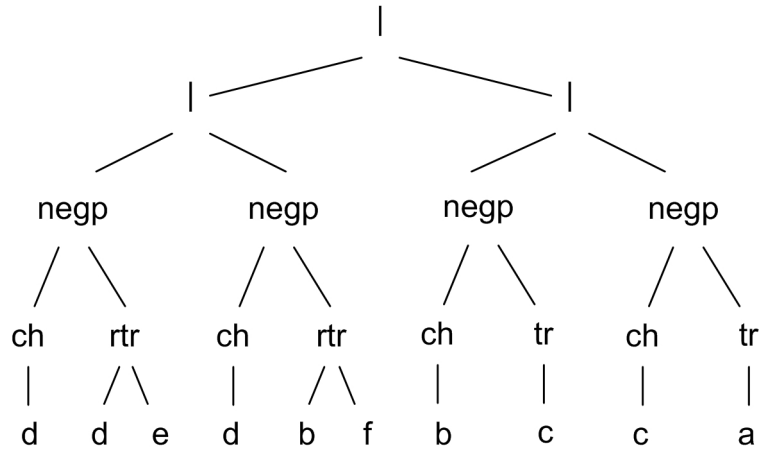


Figure 7.14: D038 Target GP Tree

7.4.5 GP Parameters and Settings

Table 7.13 lists parameters used for the D038 GP runs. Parameters were the same as for D016, except for the initial maximum and minimum tree depths, as well as the maximum tree depth applicable during the entire run. These changes in depth restrictions reflect the additional level introduced by the nested transcription factors. Similar to D016, the minimum initial tree depth corresponds to a network composed of at least 2 gates in parallel.

Table 7.13: D038 GP Parameters

population	500
maximum no. of generations	30
probability of crossover	0.9
probability of standard mutation	0.05
probability of shrink mutation	0.1
probability of reproduction	0.0
elitism	none
selection	tournament (size 3)
initial population	grow
min. initial tree depth	4
max. initial tree depth	6
maximum tree depth	9
prob. crossover point is branch	0.75
max. regenerative depth for mutation	6
max. number of retries	50
duration of SPiM simulation	100,000
no. points collected per simulation	500
no. simulations fitness score averaged over	4

7.4.6 GP Software

The software used to perform the GP runs was described previously in Section 7.2.6. A departure from that section is that the tree for the D038 target contained only the network expression and not the rates. As well, the inducer, $rep(aTc)$, was added in parallel to each candidate expression prior to the SPiM simulation to model the [with aTc, without IPTG] combination of input probes.

Typical Run-times

Run times were highly dependent on the computer system, the features in the fitness function, the number of times the expression was evaluated per fitness score, the GP parameters

(population, maximum generations), and the expressions encountered during the GP run. For the D038 target, runs were completed within 1.5 to 4.5 days. Most of the run-time was attributed to the SPiM simulations.

7.4.7 Results

The target expression was found in 8 of the 20 GP runs which were performed. With 95% confidence, this corresponds to a success rate between 22% and 61% (Appendix D). On average, the target expression was identified in the 21st generation. Median population fitness and best-of-generation fitness by generation averaged over all runs are shown in Figure 7.15. Twenty baseline runs with tournament size 1 were also completed. Among these 20 baseline runs, the target expression was never constructed.

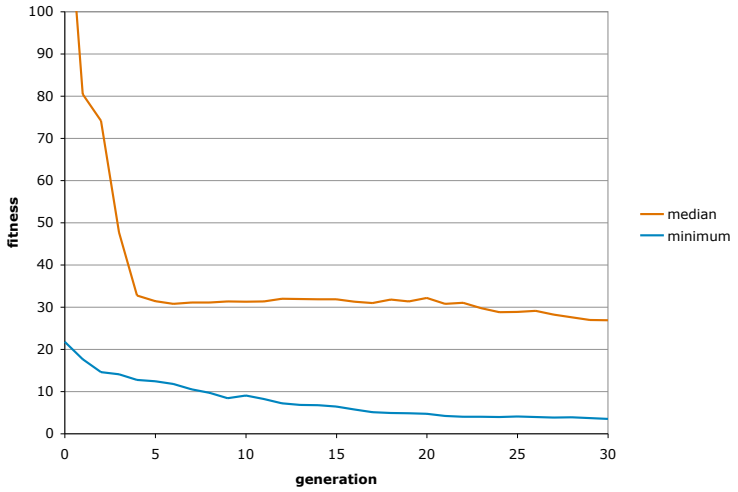


Figure 7.15: D038 GP Results Averaged over 20 Runs

7.4.8 Discussion

The GP successfully evolved the target expression for D038 in 40% of the runs. This reduction in proportion of successful runs compared to the D016 experiment can be attributed to the increased difficulty of the target, as reflected by the lower probability of randomly constructing the target tree. These positive results demonstrate the effectiveness and ability of GP to synthesize networks from the modular gene gate constructs, making use of the feature-based fitness function to guide the search.

Table 7.14: D038 Fitness Score Comparison

	Fitness		
	Minimum	Maximum	Median
Target (when found)	2.35	3.83	3.17
Best-of-run (when target not found)	2.18	9.53	2.60

The fitness function incorporated a subset of 10 features from 4 channels of information. Table 7.14 compares the range and median of fitness scores from the GP runs which did and did not find the target. Upon examination of the table and the list of the top individuals from successful runs, it is evident that there are expressions other than the target that are receiving scores comparable to the target. The addition of further features to the fitness function, perhaps from additional channels could more definitively distinguish the target from other expressions. A closer look at the behaviour of these off-target expressions would indicate which features to add.

As was done for D016, two near target expressions were examined:

1. Near target #1: negp(d, rtr(d,e)) | negp(d, tr(b)) | negp(b,tr(c)) | negp(c,tr(a))
   This expression contains 3 of the 4 gates in the target (negp(d, tr(b)) replaced negp(d, rtr(b,f))) and received a score of 2.34 which was comparable to the range of scores obtained for the target expression, 2.35 to 3.83 (Table 7.14).

2. Near target #2:
   negp(d, rtr(d,e)) | negp(d, rtr(a,e)) | negp(b,rtr(c,f)) | negp(c,rtr(a,f)) | negp(d, tr(b))
   The expression contains only 1 gate present in the target (negp(d, rtr(d,e))), and 4 additional gates. It received a fitness score of 9.53 which falls outside the range of observed D038 scores.

Simulations of these expressions are found in Figure 7.16 alongside those for the target expression for comparison purposes. There appears to be no visible difference between the target and near target #1 behaviours, however differences with near target #2's behaviour is evident. Closer study of the values of the features which contributed to the scores did not reveal any obvious differences between the target and near target #1. However, a significant difference was identified between the target and near target #2 for several features, consistent with the observations, namely in the mean (and somewhat for the standard deviation) for GFP and in the standard deviations for lcI and TetR.

As for D016, these observations confirm once again that the feature-based fitness function is capable of identifying the target behaviour and is showing that the features are effectively distinguishing between changes in behaviour.

7.4.9 Further Work

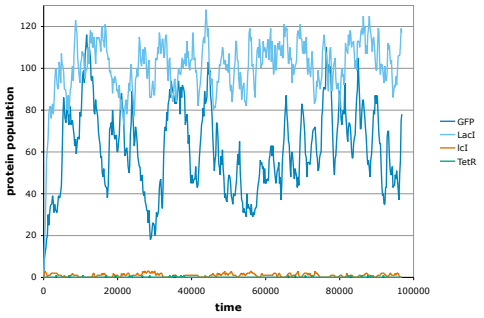
Considering that the population size was at the lower limits recommended for the GP algorithm [50] and that the target expression was found several times among the 20 runs, the fitness function could be further challenged with a more difficult system. For instance, rates could be added as parameters in the $negp$ gates and evolved.

As previously discussed, further efforts into selecting an alternate subset of features for use in the fitness function would be beneficial in order to ensure that the target can be more clearly delineated from other expressions when it is encountered in a run.

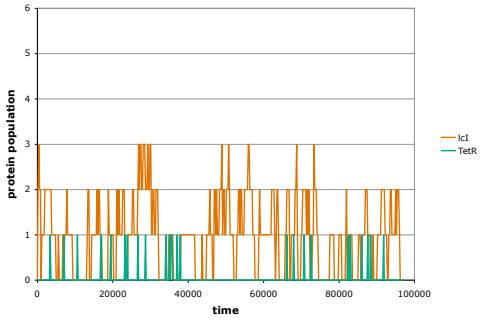
7.4.10 Supporting Documentation

The following supporting documentation is made available in the Appendices or on the accompanying DVD:

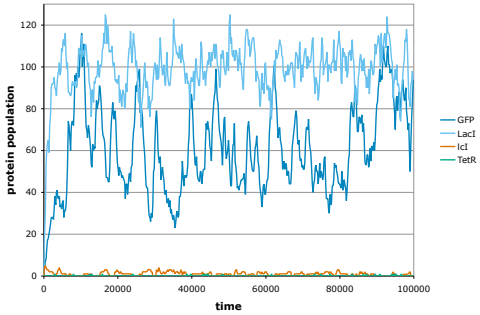
- features for 4 channels over 200 simulations
- SPiM input file for the target expression
- probability tree for random generation of the target expression
- openBeagle files tailored for D038
- output and log files for all D038 runs (including openBeagle reports, if generated)
- implementation details for the GP and fitness function
- confidence interval calculations



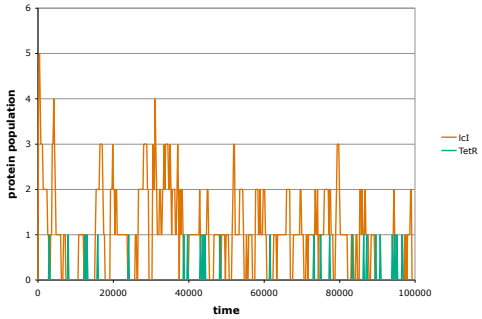
(a) Target (all channels)



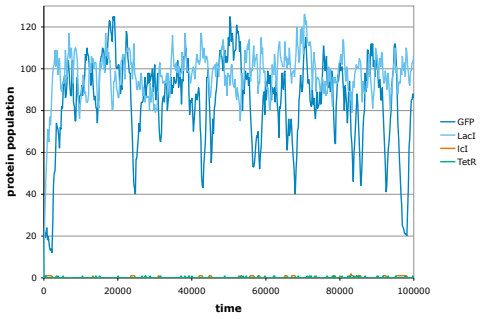
(b) Target (lcI and TetR only)



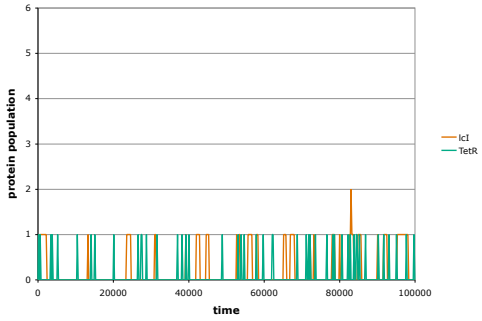
(c) Near-target #1 (all channels)



(d) Near-target #1 (lcI and TetR only)]



(e) Near-target #2 (all channels)



(f) Near-target #2 (lcI and TetR only)

Figure 7.16: SPiM Simulations of the D038 Target and Near-Target Expressions

Chapter 8

Discussion

8.1 Effectiveness of the Feature-based Fitness Function

In conjunction with the feature-based fitness function, GP successfully synthesized targeted expressions exhibiting a variety of behaviours arising from probabilistic GRN networks and symbolic regression with added noise. The only experiment where it failed to do so was for the spherical Bessel function, $j1$. Although the target expression was not evolved in this case, the GP managed, amid the noise, to produce expressions with oscillating and attenuating behaviour, very similar to $j1$'s. Examination of near target expressions obtained in the D016 and D038 experiments also demonstrated how the features could create a viable search space. These expressions, which received favourable scores, contained elements of the targeted expression. When their fitness score was comparable to that of the target expression, their behaviour and feature values were also very similar. When their fitness score was a bit off, it could be explained by both the behaviour (sometimes upon closer scrutiny) and by certain features in the subset.

Considering the success in evolving the desired behaviours and that the GP was run with rather low population sizes (ranging from 200 to 1000), there is potential for the feature-based fitness function to tackle more challenging and complex behaviours.

8.2 Fitness Function Design

The following sections discuss aspects of the feature-based fitness function design, in light of the results obtained from the multiple experiments.

8.2.1 Full Set of Features

Aside from the Bessel function experiment, the comprehensive set of 17 features proved to be sufficient to identify the target expression. There was always an ample number of features whose values were relatively stable among several evaluations of the target expression.

The trend and seasonally adjusted (tsa) characteristics were not selected for any of the fitness functions, mostly because they tended to be less stable than their counterparts based on raw data. As well, it was considered redundant to include both the raw and tsa features in the subset, and the raw feature was perceived to be more reliable.

The adequacy of the full set in these experiments does not preclude considering other features. There are numerous statistical features available and other measures which may make sense to include according to the problem at hand. The current set did not include any multivariate features, which could prove to be very effective when targeting concurrent signals, such as those encountered in the D016 and D038 experiments.

8.2.2 Selecting Features from the Full Set

The subset of features chosen for the fitness function ranged from 3 to 17 in size. For the non-oscillating symbolic regression target, 3 features was sufficient to evolve and distinctly identify the target. For the D016 network, 17 features were used in the fitness function and preliminary experiments found that 2 network combinations resulting in 8 channels of information were required to delineate the target expression from others. Ten features from 4 channels were used to target the more complex D038 network. The results from the D038 runs showed that there was overlap between the fitness ranges of the target and near target expressions.

Although stability amid the noise was the predominant criteria for choosing features for the fitness function, a more rigorous approach would be beneficial for the performance and efficiency of the search. Identifying the most effective subset falls under Feature Subset Selection, a topic which has been studied extensively [44]. This selection process need only be performed once at the beginning of each problem, so it would be a worthwhile effort to investigate. Natural extensions to feature subset selection include feature weighting within the fitness function and feature extraction, where basic features are combined via GP to create a more sophisticated, compound feature. Although these extensions may increase performance, they would also detract from the practical and easy-to-comprehend nature of the proposed fitness function.

8.2.3 Fitness Function Formula

A comparison between normalization by average and normalization by standard deviation carried out in the D016 experiments indicated that the standard deviation approach enabled the fitness function to more distinctly identify the target expression.

8.2.4 Number of Repeated Evaluations

The D016 experiment showed that a single evaluation was adequate to evolve the target. However, 4 evaluations increased the fitness function's ability to more distinctly identify the target expression. A side study documented in Appendix A performed on the non-oscillating symbolic regression problem suggested that too many evaluations may be detrimental to the search, leading to a reduction in the GP's performance.

Performing multiple evaluations is considered a rudimentary and inefficient way of dealing with noise in fitness evaluations [29]. In the gene gate experiments, it was observed that much of the run-time was spent on the SPiM simulations. One idea that would be easy to implement is to base the score on multiple evaluations only if the fitness of the first evaluation lies below a certain threshold. By doing this, promising expressions would continue to be evaluated multiple times, while reduced effort would be spent on the evaluation of less favourable individuals. Other ways to deal with the uncertainty in fitness evaluation have been studied [5] [29]. It would be worthwhile to explore this area when there is a need for increased efficiency.

Chapter 9

Conclusions

A feature-based fitness function was developed to evaluate noisy or stochastic time series in which the score was calculated as a sum-of-errors from a set of statistical features characterizing the temporal data. The set of features was drawn from a comprehensive set of 17 statistical features, preferring those which exhibited stability amid the noise. This approach produced a measure which is easy to interpret and implement, and versatile in that it could be tailored to describe a variety of behaviours.

With the use of this fitness function, a genetic programming system successfully evolved several targeted expressions in experiments involving symbolic regression with added noise, as well as modular gene regulatory network models based on the stochastic π-calculus. The targeted expressions were of varying complexity and included both oscillating and non-oscillating behaviour.

Stochastic and noisy behaviour can significantly compromise the performance of the standard fitness function approach, which involves taking the sum-of-errors directly from the values of the time series. This feature-based fitness function offers an alternative fitness measure when dealing with systems containing such uncertainties. It can be readily employed by search and optimization algorithms, providing a tool for scientists to construct and explore models which incorporate more complex, real-life phenomena.

There is plenty of further work that can be performed to explore the capabilities of this fitness function and to improve its performance:

- Develop a more rigorous feature subset selection method to identify a suitable set of features for the fitness function which optimizes its performance.

- Challenge the fitness function with problems of increased difficulty. For the stochastic gene gate model, more sophisticated models containing further biological detail, such as the delay and reaction rates, could be evolved. Another suggestion is to add automatically defined functions (ADFs) to the genetic programming system such that the stochastic π-calculus within the gene gates can be evolved concurrently with the gene gate expressions.

- Consider other features such as multivariate statistics when dealing with concurrent signals.

- Improve the efficiency in dealing with repeated evaluations by optimizing the number or implementing shortcuts so that time is not wasted on poorly performing expressions.

Bibliography

[1] Association for the Advancement of Artificial Intelligence (AAAI) Website. Accessed Oct. 2008 at `http://www.aaai.org/home.html`.

[2] R. J. Alcock and Y. Manolopoulos. Time-series similarity queries employing a feature-based approach. In *Proceeding of 7th Hellenic Conference on Informatics*, pages III.1–9, august 1999.

[3] Shin Ando, Erina Sakamoto, and Hitoshi Iba. Evolutionary modeling and inference of gene network. *Inf. Sci*, 145(3-4):237–259, 2002.

[4] C. M. Antunes and A. L. Oliveira. Temporal data mining: An overview. 2001.

[5] P. G. Balaji, D. Srinivasan, and C. K. Tham. Uncertainties reducing techniques in evolutionary computation. In Dipti Srinivasan and Lipo Wang, editors, *2007 IEEE Congress on Evolutionary Computation*, pages 556–563, Singapore, 25-28 September 2007. IEEE Computational Intelligence Society, IEEE Press.

[6] Ziv Bar-Joseph. Analyzing time series gene expression data. *Bioinformatics*, 20(16):2493–2503, 2004.

[7] Ralf Blossey, Luca Cardelli, and Andrew Phillips. A compositional approach to the stochastic dynamics of gene networks. *Trans. in Comp. Sys. Bio (TCSB*, 3939:99–122, 2006.

[8] Ralf Blossey, Luca Cardelli, and Andrew Phillips. Compositionality, stochasticity and cooperativity in dynamic models of gene regulation. *HFSP Journal*, 2(1):17–28, February 2008.

[9] A. Borrelli, I. De Falco, A. Della Cioppa, M. Nicodemi, and G. Trautteura. Performance of genetic programming to extract the trend in noisy data series. *Physica A: Statistical and Theoretical Physics*, 370(1):104–108, 1 October 2006. Econophysics Colloquium - Proceedings of the International Conference "Econophysics Colloquium".

[10] Luca Cardelli. Abstract machines of systems biology. 3737:145–168, 2005.

[11] Dong-Yeon Cho, Kwang-Hyun Cho, and Byoung-Tak Zhang. Identification of biochemical networks by S-tree based genetic programming. *Bioinformatics*, 22(13):1631–1640, 2006.

[12] Dominique Chu. Evolving genetic regulatory networks for systems biology. In Dipti Srinivasan and Lipo Wang, editors, *2007 IEEE Congress on Evolutionary Computation*, pages 875–882, Singapore, 25-28 September 2007. IEEE Computational Intelligence Society, IEEE Press.

[13] Ivanoe De Falco, Antonio Della Cioppa, Domenico Maisto, Umberto Scafuri, and Ernesto Tarantino. Parsimony doesn't mean simplicity: Genetic programming for inductive inference on noisy data. In Marc Ebner, Michael O'Neill, Anikó Ekárt, Leonardo Vanneschi, and Anna Isabel Esparcia-Alcázar, editors, *Proceedings of the 10th European Conference on Genetic Programming*, volume 4445 of *Lecture Notes in Computer Science*, pages 351–360, Valencia, Spain, 11 - 13 April 2007. Springer.

[14] Frank Dellaert, Thomas Polzin, and Alex Waibel. Recognizing emotion in speech. In *Proceedings. 4th International Conference on Spoken Language Processing, ICSLP 96*, Philadelphia, PA, 1996. IEEE.

[15] Barry Drennan and Randall D. Beer. A model for exploring genetic control of artificial amoebae. In Jordan Pollack, Mark Bedau, Phil Husbands, Takashi Ikegami, and Richard A. Watson, editors, *Artificial Life IX : Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems*, pages 381–386. MIT Press, 2004.

[16] Barry Drennan and Randall D. Beer. Evolution of repressilators using a biologically-motivated model of gene expression. In Luis M. Rocha, Larry S. Yaeger, Mark A. Bedau, Dario Floreano, Robert L. Goldstone, and Alessandro Vespignani, editors, *Artificial Life X : Proceedings of the Tenth International Conference on the Simulation and Synthesis of Living Systems*, pages 22–27. International Society for Artificial Life, The MIT Press (Bradford Books), August 2006.

[17] M. B. Elowitz and S. Leibler. A synthetic oscillatory network of transcriptional regulators. *Nature*, 403:335–338, January 2000.

[18] M. B. Elowitz, A. J. Levine, E. D. Siggia, and P. S. Swain. Stochastic Gene Expression in a Single Cell. *Science*, 297:1183–1186, August 2002.

[19] Jasmin Fisher and Thomas A. Henzinger. Executable cell biology. *Nature Biotechnology*, 25(11):1239–1249, November 2007.

[20] John Fox. *car: Companion to Applied Regression*, 2006.

[21] Chris Fraley, Fritz Leisch, Martin Maechler, Valderio Reisen, and Artur Lemonte. *fracdiff: Fractionally differenced ARIMA aka ARFIMA(p,d,q) Models*, 2006.

[22] P. François and V. Hakim. Design of genetic networks with specified functions by evolution in silico. *Proceedings of the National Academy of Science*, 101:580–585, January 2004.

[23] Christian Gagné and Marc Parizeau. Genericity in evolutionary computation software tools: Principles and case-study. *International Journal on Artificial Intelligence Tools*, 15(2):173–194, 2006.

[24] Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.*, 81(25):2340–2361, 1977.

[25] Calin C. Guet, Michael B. Elowitz, Weihong Hsing, and Stanislas Leibler. Combinatorial synthesis of genetic networks. *Science*, 296(5572):1466–1470, 2002.

[26] Robert C. Hilborn. *Chaos and nonlinear dynamic: an introduction for scientists and engineers*. Oxford University Press, 1994.

[27] Mark P. Hinchliffe and Mark J. Willis. Dynamic systems modelling using genetic programming. *Computers & Chemical Engineering*, 27(12):1841–1854, 2003.

[28] Janine H. Imada and Brian J. Ross. Using feature-based fitness evaluation in symbolic regression with added noise. In Marc Ebner, Mike Cattolico, Jano van Hemert, Steven Gustafson, Laurence D. Merkle, Frank W. Moore, Clare Bates Congdon, Christopher D. Clack, Frank W. Moore, William Rand, Sevan G. Ficici, Rick Riolo, Jaume Bacardit, Ester Bernado-Mansilla, Martin V. Butz, Stephen L. Smith, Stefano Cagnoni, Mark Hauschild, Martin Pelikan, and Kumara Sastry, editors, *GECCO-2008 Late-Breaking Papers*, pages 2153–2158, Atlanta, GA, USA, 12-16 July 2008. ACM.

[29] Yaochu Jin and Jürgen Branke. Evolutionary optimization in uncertain environments-a survey. *IEEE Trans. Evolutionary Computation*, 9(3):303–317, 2005.

[30] Yaochu Jin and Bernhard Sendhoff. Evolving in silico bistable and oscillatory dynamics for gene regulatory network motifs. In Jun Wang, editor, *2008 IEEE World Congress on Computational Intelligence*, Hong Kong, 1-6 June 2008. IEEE Computational Intelligence Society, IEEE Press.

[31] Mohammed Waleed Kadous. Learning comprehensible descriptions of multivariate time series. In *Proc. 16th International Conf. on Machine Learning*, pages 454–463. Morgan Kaufmann, San Francisco, CA, 1999.

[32] Shinichi Kikuchi, Daisuke Tominaga, Masanori Arita, Katsutoshi Takahashi, and Masaru Tomita. Dynamic modeling of genetic networks using genetic algorithm and S-system. *Bioinformatics*, 19(5):643–650, 2003.

[33] J. Kitagawa and H. Iba. Identifying metabolic pathways and gene regulation networks with evolutionary algorithms. In G. B. Fogel and D. W. Corne, editors, *Evolutionary Computing in Bioinformatics.*, chapter 12. MorganKaufmann, September 2002.

[34] Johannes F. Knabe, Chrystpoher L. Nehaniv, Maria J. Schilstra, and Tom Quick. Evolving biological clocks using genetic regulatory networks. In Luis M. Rocha, Larry S. Yaeger, Mark A. Bedau, Dario Floreano, Robert L. Goldstone, and Alessandro Vespignani, editors, *Artificial Life X : Proceedings of the Tenth International Conference on the Simulation and Synthesis of Living Systems*, pages 15–21. International Society for Artificial Life, The MIT Press (Bradford Books), August 2006.

[35] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[36] John R. Koza, William Mydlowec, Guido Lanza, Jessen Yu, and Martin A. Keane. Automatic computational discovery of chemical reaction networks using genetic programming. In Saso Dzeroski and Ljupco Todorovski, editors, *Computational Discovery of Scientific Knowledge*, volume 4660 of *Lecture Notes in Computer Science*, pages 205–227. Springer, 2007.

[37] Pedro Larrañaga, Borja Calvo, Roberto Santana, Concha Bielza, Josu Galdiano, Iñaki Inza, José Antonio Lozano, Rubén Armañanzas, Guzmán Santafé, Aritz Pérez Martínez, and Victor Robles. Machine learning in bioinformatics. *Briefings in Bioinformatics*, 7(1):86–112, 2006.

[38] K. Lavangnananda and A. Piyatumrong. Image processing approach to features extraction in classification of control chart patterns. In *2005 IEEE Mid-Summer Workshop on Soft Computing in Industrial Applications (SMCia/05)*, pages 85–90, 2005.

[39] Srivatsan Laxman and P. S. Sastry. A survey of temporal data mining. *SADHANA: Academy Proceeding in Engineering Sciences*, 31(2):173–198, April 2006. Special Issue on Statistical Techniques in Electrical and Computer Engineering.

[40] André Leier and Kevin Burrage. Evolving genetic regulatory networks performing as stochastic switches. In T. Kovacs and A. R. Marshall, editors, *Artificial Intelligence and Simulation of Behaviour (AISB) Conference 2006: Adaptation in Artificial and Biological Systems*, volume 3, pages 150–157. Society for the Study of AI and Simulation of Behaviour, 2006.

[41] André Leier, P. Dwight Kuo, Wolfgang Banzhaf, and Kevin Burrage. Evolving noisy oscillatory dynamics in genetic regulatory networks. In Pierre Collet, Marco Tomassini, Marc Ebner, Steven Gustafson, and Anikó Ekárt, editors, *EuroGP*, volume 3905 of *Lecture Notes in Computer Science*, pages 290–299. Springer, 2006.

[42] Benjamin Lewin. *Genes IX*. Jones and Bartlett, 2008.

[43] T. W. Liao. Clustering of time series data: A survey. *Pattern Recognition*, 38(11):1857–1874, November 2005.

[44] Huan Liu and Hiroshi Motoda. *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[45] Heitor S. Lopes. Genetic programming for epileptic pattern recognition in electroencephalographic signals. *Appl. Soft Comput.*, 7(1):343–352, 2007.

[46] Spryros Makridakis, Steven C. Wheelwright, and Rob J. Hyndman. *Forecasting Methods and Applications*. John Wiley & Sons Inc., third edition, 1998.

[47] David S. Moore. *The Basic Practice of Statistics*. W. H. Freeman and Company, third edition, 2004.

[48] Alex Nanopoulos, Rob Alcock, and Yannis Manolopoulos. Feature-based classification of time-series data. In *Information processing and technology*, pages 49–61. Nova Science Publishers, Inc., Commack, NY, USA, 2001.

[49] Andrew Phillips. The Stochastic Pi Machine (SPiM). Accessed at `http://research.microsoft.com/ aphillip/spim/`, 2008.

[50] Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. (With contributions by J. R. Koza).

[51] Lijun Qian, Haixin Wang, and Edward R. Dougherty. Inference of noisy nonlinear differential equation models for gene regulatory networks using genetic programming and kalman filtering. *IEEE Transactions on Signal Processing*, 56(7):3327–3339, July 2008.

[52] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2007. ISBN 3-900051-07-0.

[53] K. Rodriguez-Vazquez and P. J. Fleming. Evolution of mathematical models of chaotic systems based on multiobjective genetic programming. *Knowledge and Information Systems*, 8(2):235–256, August 2005.

[54] K. Rodríguez-Vázquez, C. M. Fonseca, and P. J. Fleming. Identifying the Structure of NonLinear Dynamic Systems Using Multiobjective Genetic Programming. *IEEE Transactions on Systems, Man, and Cybernetics—Part A: Systems and Humans*, 34(4):531–545, July 2004.

[55] B.J. Ross. Using genetic programming to synthesize monotonic stochastic processes. In R. Andonie, editor, *Computational Intelligence (CI 2007)*, pages 71–78, 2007.

[56] Erina Sakamoto and Hitoshi Iba. Inferring a system of differential equations for a gene regulatory network by using genetic programming. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, pages 720–726, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, 27-30 May 2001. IEEE Press.

[57] Roy Schwaerzel and Tom Bylander. Predicting currency exchange rates by genetic programming with trigonometric functions and high-order statistics. In Mike Cattolico, editor, *Genetic and Evolutionary Computation Conference, GECCO 2006, Proceedings, Seattle, Washington, USA, July 8-12, 2006*, pages 955–956. ACM, 2006.

[58] Sara Silva and Yao-Ting Tseng. Classification of seafloor habitats using genetic programming. In Franz Rothlauf, editor, *Late breaking paper at Genetic and Evolutionary Computation Conference (GECCO'2005)*, Washington, D.C., USA, 25-29 June 2005.

[59] Felix Streichert, Hannes Planatscher, Christian Spieth, Holger Ulmer, and Andreas Zell. Comparing genetic programming and evolution strategies on inferring gene regulatory networks. In Kalyanmoy Deb, Riccardo Poli, Wolfgang Banzhaf, Hans-Georg Beyer, Edmund Burke, Paul Darwen, Dipankar Dasgupta, Dario Floreano, James Foster, Mark Harman, Owen Holland, Pier Luca Lanzi, Lee Spector, Andrea Tettamanzi, Dirk Thierens, and Andy Tyrrell, editors, *Genetic and Evolutionary Computation – GECCO-2004, Part I*, volume 3102 of *Lecture Notes in Computer Science*, pages 471–480, Seattle, WA, USA, 26-30 June 2004. Springer-Verlag.

[60] Ruixiang Sun, Fugee Tsung, and Liangsheg Qu. Combining bootstrap and genetic programming for feature discovery in diesel engine diagnosis. *International Journal of Industrial Engineering*, 11(3):273–281, 2004.

[61] Jochen Supper, Holger Fröhlich, Christian Spieth, Andreas Dräger, and Andreas Zell. Inferring gene regulatory networks by machine learning methods. In David Sankoff, Lusheng Wang, and Francis Chin, editors, *APBC*, volume 5 of *Advances in Bioinformatics and Computational Biology*, pages 247–256. Imperial College Press, 2007.

[62] Timo Teräsvirta, Chien-Fu Lin, and Clive W. J. Granger. Power of the neural network linearity test. *Journal of Time Series Analysis*, 14(2):209–220, 1993.

[63] Gasper Tkačik and William Bialek. Cell biology: Networks, regulation, pathways. 2007.

[64] Adrian Trapletti and Kurt Hornik. *tseries: Time Series Analysis and Computational Finance*, 2007.

[65] Berwin A. Turlach and Andreas Weingessel. *quadprog: Functions to solve Quadratic Programming Problems*, 2007.

[66] Xiaozhe Wang. *Characteristic-based Forecasting for Time Series Data*. PhD thesis, Monash University, 2005.

[67] Xiaozhe Wang, Kate Smith, and Rob Hyndman. Characteristic-based clustering for time series data. *Data Min. Knowl. Discov.*, 13(3):335–364, 2006.

[68] Xiaozhe Wang, Anthony Wirth, and Liang Wang. Structure-based statistical features and multivariate time series clustering. In *ICDM*, pages 351–360. IEEE Computer Society, 2007.

[69] Walter Willinger, Vern Paxson, and Murad S. Taqqu. Self-similarity and heavy tails: Structural modeling of network traffic. In *Statistical Techniques and Applications*, pages 27–53. Verlag, 1998.

[70] Achim Zeileis and Gabor Grothendieck. *zoo: S3 Infrastructure for Regular and Irregular Time Series*, 2007.

Appendix A

Study on the Number of Repeated Evaluations

A.1 Introduction

In many of the symbolic regression and gene gate experiments, candidate expressions were evaluated 4 times and the average of the 4 resulting fitnesses served as the overall score for that particular individual. A small study was performed in which the number of repeated evaluations was varied in order to examine the effect on GP performance. This experiment was carried out on the symbolic regression non-oscillating expression, $x^4 + x^3 + x^2 + x$, with added noise evaluated through the interval $[-1, 1]$. The GP was run 20 times for the following number of repeated evaluations: 1, 2, 4, 6, 10.

A.2 Experimental Settings

On the most part, the experiment followed the same approach as was taken for the Symbolic Regression experiments described in Chapter 6. The most significant departures were that the targeted feature values were based on those with added noise (as opposed to without added noise), and 5 features were used in the fitness function (as opposed to 3). The experimental settings are outlined in the following subsections. Refer to Chapter 6 for further details.

A.2.1 Added Noise

Gaussian noise, $g(0, 0.2)$, was added to each point at which a candidate expression was evaluated. This level of noise corresponded to 5% of the range of target values within the

Table A.1: Features used in the Fitness Function

no.	feature	average [a]	standard deviation	inverse coeff. of variation
1	mean	0.534	0.014	38.2
2	standard deviation	1.122	0.014	79.7
3	skew	1.416	0.040	35.7
4	serial correlation	11.064	0.175	63.2
5	self-similarity	1.000	0.000	23996

[a] increased precision was used in GP runs

Table A.2: GP Function and Terminal Sets for the Non-Oscillating Target

function set	terminal set
$+, -, *, \%,$ sin, cos, exp, ln	x

interval considered. The target features were also based on the average values over multiple evaluations of the target expression with this level of noise added.

A.2.2 Fitness Function

Each candidate expression was evaluated at 201 evenly-spaced points over the interval. After the features were calculated from the resulting set of values, the fitness was calculated based on equation 5.1 in Section 5.2, normalized by the average target feature values.

The fitness function was based on 5 features (mean, standard deviation, skew, serial correlation and self-similarity). This subset was chosen by examining the features from 500 evaluations of the target function with noise added. Those with high inverse coefficients of variation were selected. As well, the target feature values were determined from this set of 500 interpretations. Table A.1 lists the selected features along with their average, standard deviation and inverse coefficient of variation values with $g(0, 0.2)$ added noise.

A.2.3 GP Function and Terminal Sets

The function and terminal sets remained unchanged as listed in Table A.2.

A.2.4 GP Parameters and Settings

The same set of parameter settings were applied. They are reiterated in Table A.3.

Table A.3: GP Parameters

population	500
maximum no. of generations	20
probability of crossover	0.9
probability of mutation	0.1
probability of reproduction	0.0
elitism	none
selection	tournament (size 3)
initial population	ramped half and half
min. initial tree depth	2
max. initial tree depth	6
maximum tree depth	17
prob. crossover point is branch	0.9
max. regenerative depth for mutation	5
max. number of retries	50

Table A.4: GP Results

No. of Repeated Evaluations	Number of Runs Target Found (of 20)
1	7
2	7
4	10
6	7
10	3

A.2.5 GP Software

Refer to Section 6.7 for details on the GP software used to perform these runs.

A.3 Results

Twenty runs per configuration were executed and the results are listed in Table A.4. Supporting documentation is included on the accompanying DVD.

A.4 Discussion

The best performance (10 hits in 20 runs) was obtained when the fitness was averaged over 4 evaluations. Surprisingly, the number of hits dropped to only 3 in 20 runs when the number of repeated evaluations was increased to 10.

The confidence interval for comparing these two proportions was determined using the four plus method which provides accurate results for small samples [47]. The analysis found, with 95% confidence, that 4 repeated evaluations yields $31.8 \pm 26.4\%$ more hits than 10 repeated evaluations. This calculation is documented in Figure A.1.

These results suggest that increasing the number of repeated evaluations does not necessarily translate to improvements in GP performance, and that some amount of noise may actually be helping the search. Similar observations have also been discussed in two survey papers [5] [29].

population	no. repeated evaluations	sample size	count of successes	plus four sample proportion
1	4	$n_1 = 20 + 2 = 22$	$10 + 1 = 11$	$\hat{p}_1 = 11/22$
2	10	$n_2 = 20 + 2 = 22$	$3 + 1 = 4$	$\hat{p}_2 = 4/22$

$$\begin{aligned}\text{standard error } SE &= \sqrt{\frac{\hat{p}_1\left(1-\hat{p}_1\right)}{n_1} + \frac{\hat{p}_2\left(1-\hat{p}_2\right)}{n_2}} \\ &= \sqrt{\frac{\left(\frac{11}{22}\right)\left(\frac{11}{22}\right)}{22} + \frac{\left(\frac{4}{22}\right)\left(\frac{18}{22}\right)}{22}} \\ &= 0.1346\end{aligned}$$

$$\begin{aligned}\text{plus four 95\% confidence interval is } &(\hat{p}_1 - \hat{p}_2) \pm z^* SE \\ &= \left(\frac{11}{22} - \frac{4}{22}\right) \pm (1.960)\,(0.1346) \\ &= 0.318 \pm 0.264 \\ &= 0.054 \text{ to } 0.582\end{aligned}$$

where z^* is the critical value of the standard Normal distribution corresponding to a 95% confidence level

With 95% confidence, the difference in proportions, $(p_1 - p_2)$, is 5.4% to 58.2% where p_1 is the proportion of hits among runs with 4 repeated evaluations and p_2 is the proportion of hits among runs with 10 repeated evaluations

Figure A.1: Plus Four Confidence Interval Calculation

Appendix B

Supporting Documentation for the Gene Gate Experiments

B.1 Probability Trees

Given the GP function and terminal sets, along with the minimum initial tree depth, the probability of randomly constructing the target expression in the initial population was determined. This section documents the calculations for the three targeted gene gate expressions.

B.1.1 Repressilator

The GP function and terminal sets described in Table 7.2 are reiterated below in a slightly different format (with rates expressed as powers of 10):

root ➔ (gate, rates)
rates ➔ (δ, r, η)
gate ➔ | *or* neg
| ➔ (gate, gate)
neg ➔ (ch, ch)
ch ➔ a *or* b *or* c
δ ➔ -3 *or* -4
r ➔ 0 *or* 1 *or* 2 *or* 3 *or* 4 *or* 5
η ➔ -6 *or* -5 *or* -4 *or* -3 *or* -2 *or* -1 *or* 0

Based on these sets, the repressilator target expression and rates were as follows:

$$\text{neg(a,b)} \mid \text{neg(b, c)} \mid \text{neg(c,a)} \text{ with rates } (-4,\ 2 \text{ and } -3).$$

A minimum initial tree depth of 3 (Table 7.3) corresponds to a minimum of a single neg gate in the expression. The probability of obtaining the expression branch of the target tree is depicted in Figure B.1. The probability of obtaining the rate branch of the target tree is determined in Figure B.2.

Combining the probabilities determined in Figures B.1 and B.2, yields the following overall probabilities for the following categories per Section 7.2.7:

1. **Category 1: Target expression and correct set of rates**
   probability $= \frac{1}{972} * \frac{1}{84} = \frac{1}{81{,}648}$
2. **Category 2: Target expression found, set of rates are off-target, but fitnesses are indiscernible from the target**
   probability $= \frac{1}{972} * \frac{9}{84} = \frac{1}{9{,}072}$
3. **Categories 1 and 2 Combined**
   probability $= \frac{1}{972} * \frac{10}{84} \approx \frac{1}{8{,}165}$

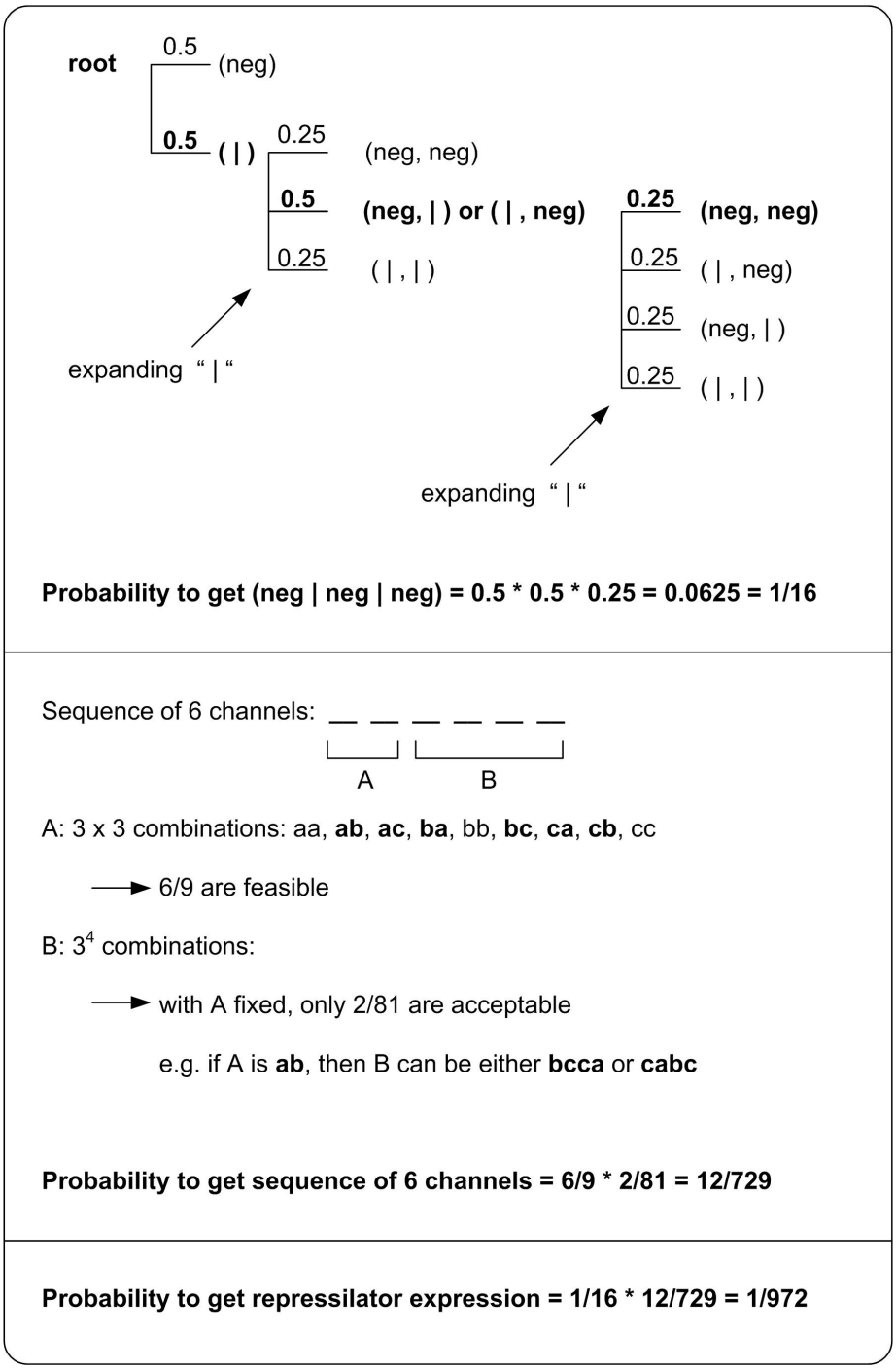


Figure B.1: Probability of Obtaining the Repressilator Expression Branch

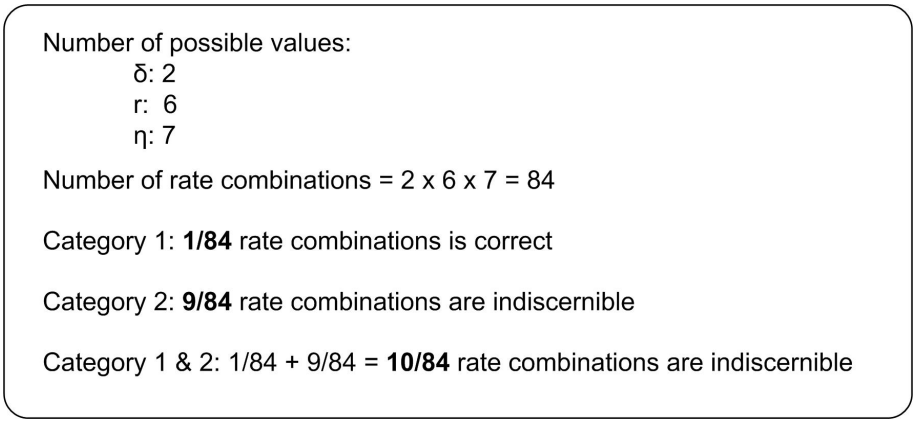


Figure B.2: Probability of Obtaining the Repressilator Rate Branch

B.1.2 D016

The GP function and terminal sets described in Table 7.6 are reiterated below in a slightly different format:

root ➔ gate
gate ➔ | *or* neg *or* negpe *or* negpf
| ➔ (gate, gate)
neg ➔ (ch, ch)
negpe ➔ ch
negpf ➔ ch
ch ➔ a *or* b *or* c *or* d

Based these sets, the D016 target expression was as follows:

$$\text{negpe(d)} \mid \text{negpf(b)} \mid \text{neg(b,c)} \mid \text{neg(c,a)}$$

A minimum initial tree depth of 3 (Table 7.7) corresponds to a minimum of a two $neg(pe/pf)$ gates in the expression. The probability of obtaining the target tree is $\frac{1}{139{,}810}$ as determined in Figures B.3 and B.4.

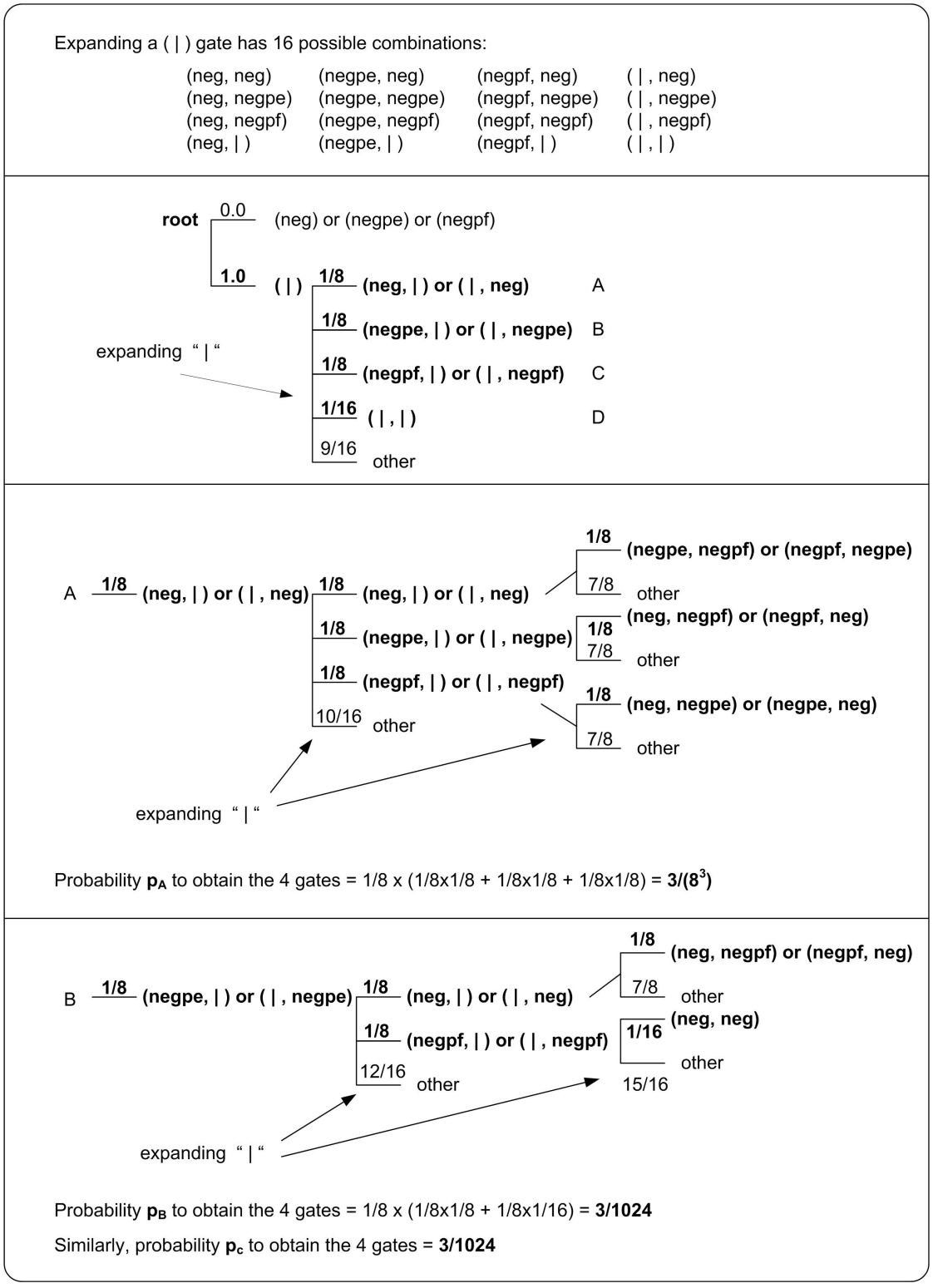


Figure B.3: Probability of Obtaining the D016 Target Tree (Part 1)

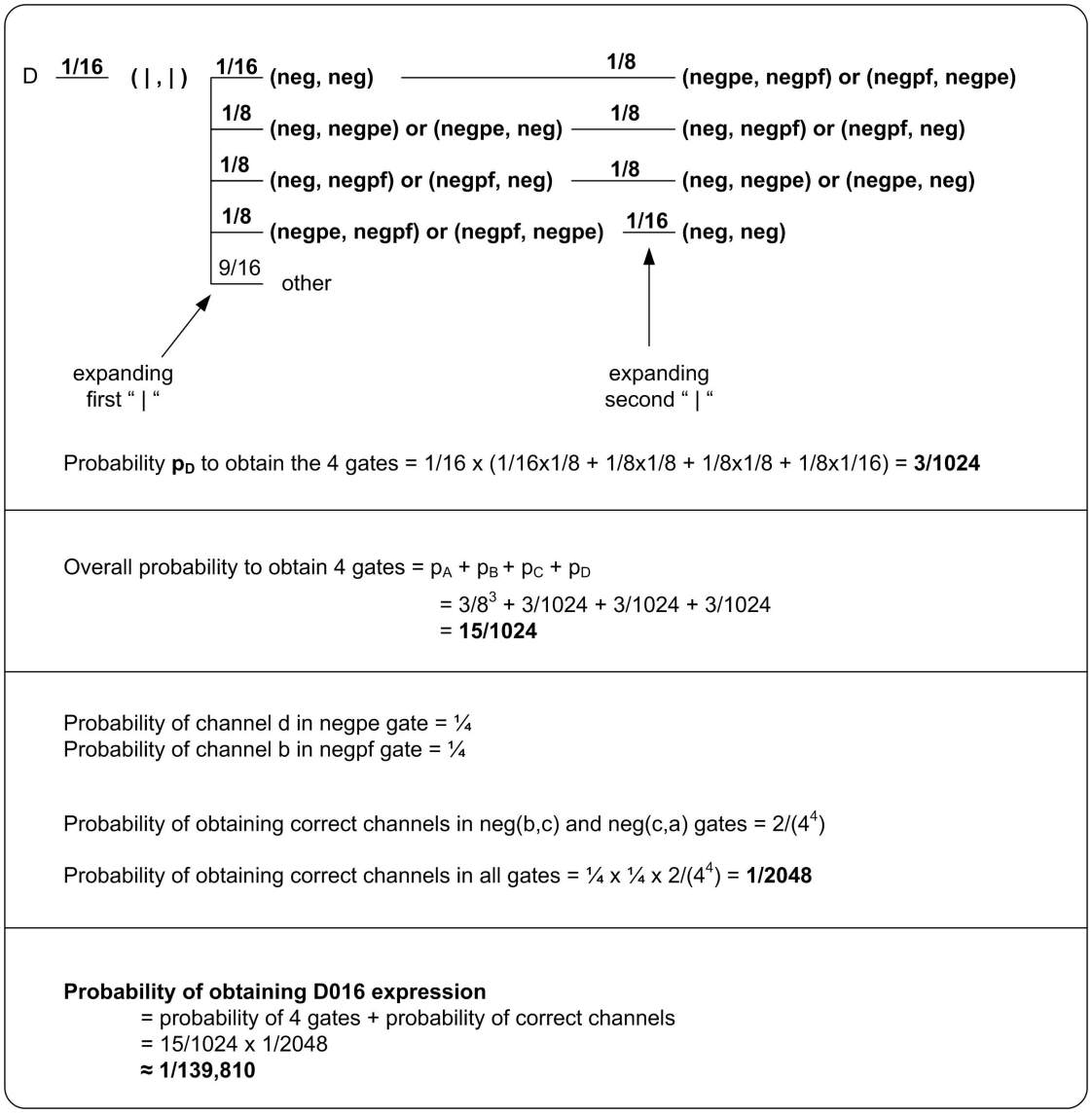


Figure B.4: Probability of Obtaining the D016 Target Tree (Part 2)

B.1.3 D038

The GP function and terminal sets described in Table 7.11 are reiterated below in a slightly different format:

root ➔ gate
gate ➔ | *or* negp
| ➔ (gate, gate)
negp ➔ (ch, tf)
tf ➔ rtr *or* tr
rtr ➔ (ch, ind)
tr ➔ (ch)
ch ➔ a *or* b *or* c *or* d
ind ➔ e *or* f

Based on these sets, the D038 target expression was as follows:

negp(d, rtr(d,e)) | negp(d, rtr(b,f)) | negp(b,tr(c)) | negp(c,tr(a))

A minimum initial tree depth of 4 (Table 7.13) corresponds to a minimum of a two $negp$ gates in the expression. The probability of obtaining the target tree is $\frac{1}{2{,}236{,}962}$ as calculated in Figure B.5.

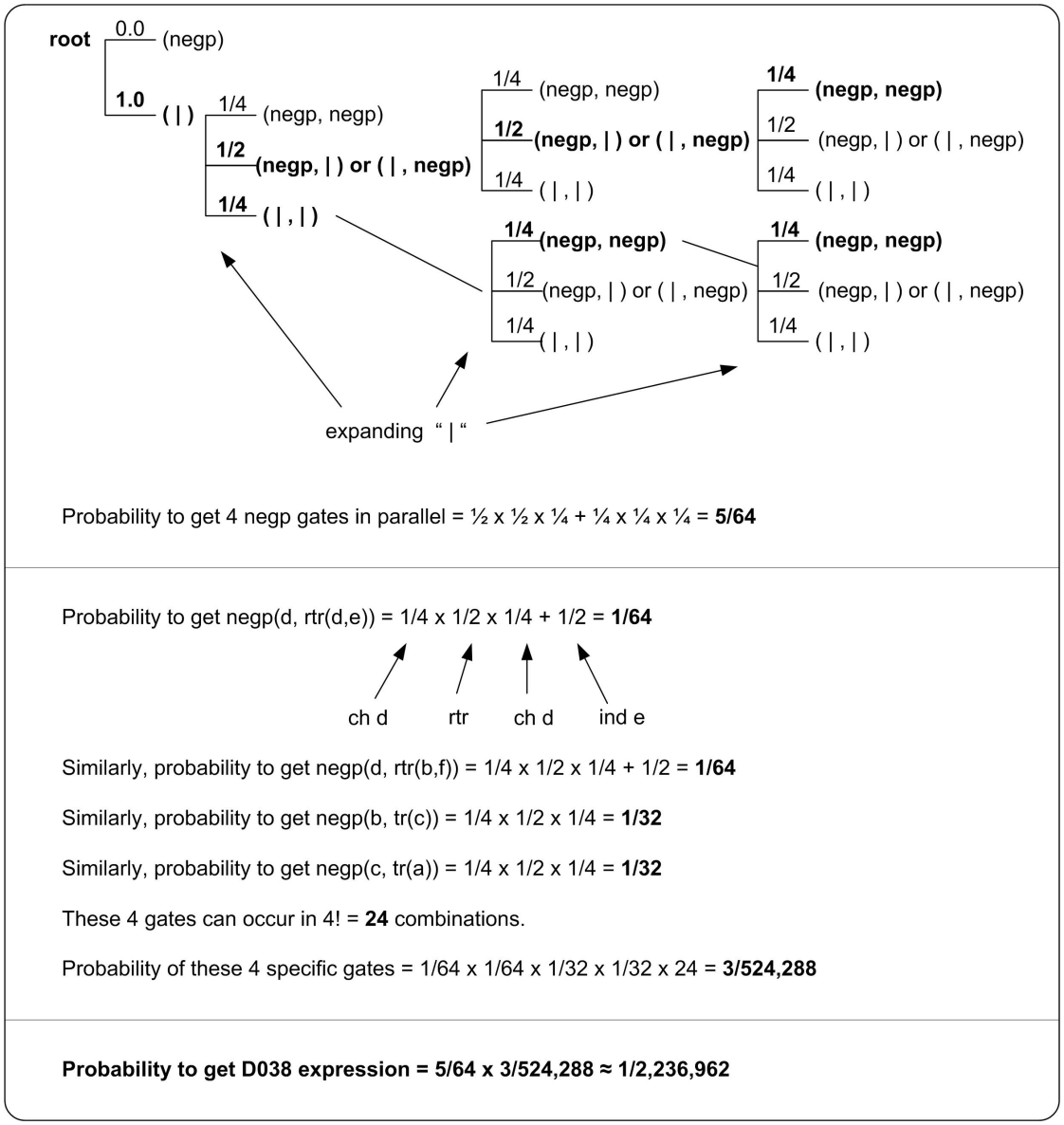


Figure B.5: Probability of Obtaining the D038 Target Tree

B.2 SPiM Input Files for the Target Expressions

B.2.1 Repressilator

```
directive sample 2000000.0 1000
directive plot !a as "a"
val e=0.1
val d=0.000100
val r=100.0000
val h=0.00100
new a @ r: chan
new b @ r: chan
new c @ r: chan
let tr(b:chan) =
  do !b; tr(b)
  or delay@d
let neg(a:chan, b:chan) =
  do delay@e; (tr(b) | neg(a,b))
  or ?a; neg_(a,b)
and neg_(a:chan,b:chan) = delay@h; neg(a,b)
run(neg(a,b)|neg(b,c)|neg(c,a))
```

B.2.2 D016

Without Inducers

```
(* Simulation time, samples, and plotting *)
directive sample 200000.0 500
directive plot !a as "a"; !b as "b"; !c as "c"; !d as "d"

(*rates*)
val dr = 0.001
val er = 0.1
val hr = 0.01

(* Transcription factor *)
let tr(b:chan()) =
do !b; tr(b)
or delay@dr

(* Repressible transcription factor *)
let rtr(a:chan(), b:chan()) =
do !a; rtr(a, b)
or !b
or delay@dr

(* Repressor *)
let rep(r:chan()) =
?r; rep(r)

(* Neg gate *)
let neg(a:chan(), b:chan()) =
do ?a; delay@hr; neg(a,b)
or delay@er; (tr(b) | neg(a,b))

(* Negp gate *)
let negp(a:chan(), b:chan()) =
do ?a; delay@hr; negp(a,b)
or delay@er; (rtr(a,b) | negp(a,b))

(* Wiring *)
new a @1.0: chan() (* GFP protein *)
new b @1.0: chan() (* LacI protein *)
new c @1.0: chan() (* lcI protein *)
new d @1.0: chan() (* TetR protein *)
new e @100.0: chan() (* aTc inducer *)
new f @100.0: chan() (* IPTG inducer *)

run ( negp(d,e) | negp(b,f) | neg(b,c) | neg(c,a) )
```

With IPTG Inducer

```
(* Simulation time, samples, and plotting *)
directive sample 100000.0 500
directive plot !a as "a"; !b as "b"; !c as "c"; !d as "d"

(*rates*)
val dr = 0.001
val er = 0.1
val hr = 0.01

(* Transcription factor *)
let tr(b:chan()) =
do !b; tr(b)
or delay@dr

(* Repressible transcription factor *)
let rtr(a:chan(), b:chan()) =
do !a; rtr(a, b)
or !b
or delay@dr

(* Repressor *)
let rep(r:chan()) =
?r; rep(r)

(* Neg gate *)
let neg(a:chan(), b:chan()) =
do ?a; delay@hr; neg(a,b)
or delay@er; (tr(b) | neg(a,b))

(* Negp gate *)
let negp(a:chan(), b:chan()) =
do ?a; delay@hr; negp(a,b)
or delay@er; (rtr(a,b) | negp(a,b))

(* Wiring *)
new a @1.0: chan() (* GFP protein *)
new b @1.0: chan() (* LacI protein *)
new c @1.0: chan() (* lcI protein *)
new d @1.0: chan() (* TetR protein *)
new e @100.0: chan() (* aTc inducer *)
new f @100.0: chan() (* IPTG inducer *)

run (negp(d,e) | negp(b,f) | neg(b,c) | neg(c,a) | rep(f))
```

B.2.3 D038

```
(* Simulation time, samples, and plotting *)
directive sample 100000.0 500
directive plot !a as "a"; !b as "b"; !c as "c"; !d as "d"

(* Degradation rate *)
val dr = 0.001

(* Transcription factor *)
let tr(b:chan()) =
do !b; tr(b)
or delay@dr

(* Repressible transcription factor *)
let rtr(b:chan(), r:chan()) =
do !b; rtr(b,r)
or !r
or delay@dr

(* Repressor *)
let rep(r:chan()) =
?r; rep(r)

(* Negp gate *)
let negp(a:chan(), p:proc(), (er:float, hr:float)) =
do ?a; delay@hr; negp(a,p, (er,hr))
or delay@er; (p() | negp(a, p, (er,hr)))

(* Wiring *)
new a @1.0: chan() (* GFP protein *)
new b @1.0: chan() (* LacI protein *)
new c @1.0: chan() (* lcI protein *)
new d @1.0: chan() (* TetR protein *)
new e @100.0: chan() (* aTc inducer *)
new f @100.0: chan() (* IPTG inducer *)

(* Auxiliary definitions: negp products *)
let rtrae() = rtr(a,e)
let rtraf() = rtr(a,f)
let rtrbe() = rtr(b,e)
let rtrbf() = rtr(b,f)
let rtrce() = rtr(c,e)
let rtrcf() = rtr(c,f)
let rtrde() = rtr(d,e)
let rtrdf() = rtr(d,f)

let tra() = tr(a)
let trb() = tr(b)
let trc() = tr(c)
let trd() = tr(d)

(* D038 Circuit *)
val r1 = (0.1, 0.25) (* rtr production and inhibition rates *)
val r2 = (0.1, 1.0) (* tr production and inhibition rates *)

run
( negp(d,rtrde,r1) | negp(d,rtrbf,r1) | negp(b,trc,r2) | negp(c,tra,r2)
| rep(e) )
```

B.3 Repressilator: Determination of the "Indiscernible" Rate Combinations

This section documents the data and statistical tests which identified the 9 "indiscernible" rate combinations for the repressilator target. These rate combinations were deemed "indiscernible" because their fitnesses could not be distinguished from those of the target set of rates.

B.3.1 Data

In conjunction with the target expression, $neg(a, b) \mid neg(b, c) \mid neg(c, a)$, 200 simulations were performed for each rate combination which typically received fitness scores below 7. The average and standard deviation of the resulting scores are found in Table B.1.

Table B.1: Fitness Statistics for Repressilator Rate Combinations

	δ	r	η	avg. fitness	std. dev.	test statistic, t
target	**-4**	**2**	**-3**	2.182996	0.794591	---
indiscernible	-4	4	-2	2.034028	0.742490	1.937
	-4	5	-3	2.056659	0.763074	1.622
	-4	3	-3	2.104634	0.838339	0.959
	-4	3	-2	2.115993	0.928780	0.775
	-4	4	-1	2.184693	0.832478	-0.021
	-4	4	-3	2.187037	0.818736	-0.050
	-4	1	-3	2.208290	0.865552	-0.304
	-4	5	-2	2.250351	0.871066	-0.808
	-4	5	-1	2.285047	0.874062	-1.222
discernible	-4	2	-2	2.447590	0.912708	-3.092
	-4	3	-1	2.560408	0.959892	-4.283
	-4	5	0	3.165100	1.166068	-9.843
	-4	4	0	3.309882	1.083719	-11.859
	-4	0	-4	4.036522	1.474576	-15.649
	-4	5	-4	4.302437	1.563078	-17.094
	-4	1	-4	4.453242	1.501185	-18.903
	-4	4	-4	4.520014	1.412630	-20.392
	-4	3	-4	4.522555	1.528296	-19.208
	-4	2	-4	4.549031	1.525407	-19.454
	-4	2	-1	5.842015	1.768171	-26.694
	-4	1	-2	5.853329	1.720617	-27.388
	-4	0	-3	6.009476	1.744902	-28.224
	-4	3	0	6.662919	1.597237	-35.514

B.3.2 Statistical Tests

The test statistic, t, for the two sample t-test between the target and each rate combination can be found in the far right column of Table B.1. This statistic was calculated according to the following formula:

$$t = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

where subscript 1 refers to the target

subscript 2 refers to the rate combination being compared to the target

$\bar{x}$ is the average fitness

s is the standard deviation of the fitness

n is the sample size

Based on this sample size, there are $(200 - 1) = 199$ degrees of freedom. The critical values, t^*, for 100 degrees of freedom from the t distribution chart for the corresponding two-sided confidence levels are as follows:

confidence level	90%	95%	99%	99.5%	99.8%
t^*	1.660	1.984	2.626	2.871	3.174

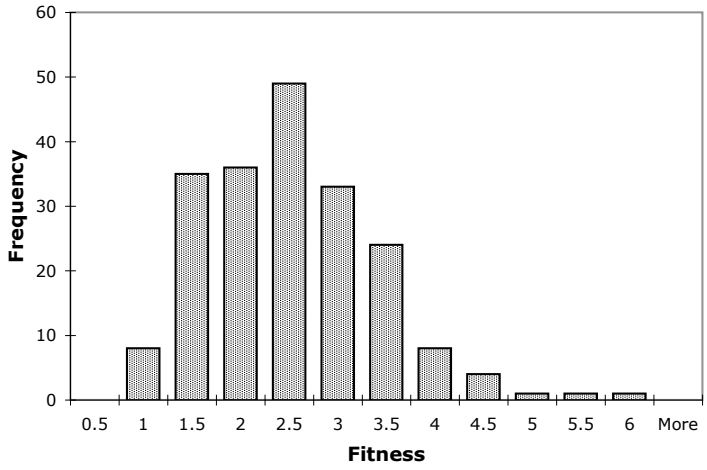
With a high level of confidence, any rate combination with a test statistic whose absolute value exceeds 3 is considered "discernible" from the target. The remaining 9 combinations are deemed "indiscernible".

To validate the use of the two sample t-test, histograms illustrating the distribution of the 200 sample fitnesses were drawn up (Figure B.6) for the target and the 2 rate combinations on either side of the "discernible" dividing line.

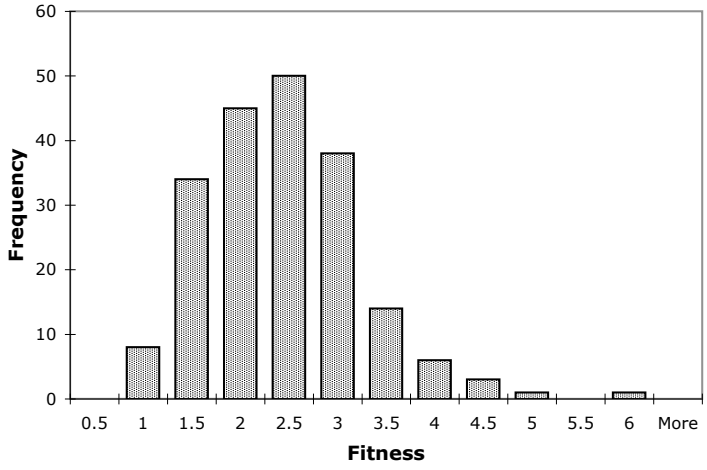
In all cases, the distribution can be described as skewed with no strong outliers. The following are guidelines which address non-normal distributions as recommended by Moore [47] in order to legitimately perform the two-sample t-test:

- Although the method is based on normally distributed populations, it is adequate for the distributions to have similar shapes and no strong outliers.
- If the distributions have different shapes, then larger samples are required.
- If the distributions are clearly skewed, use a sum of the sample sizes ≥ 40.

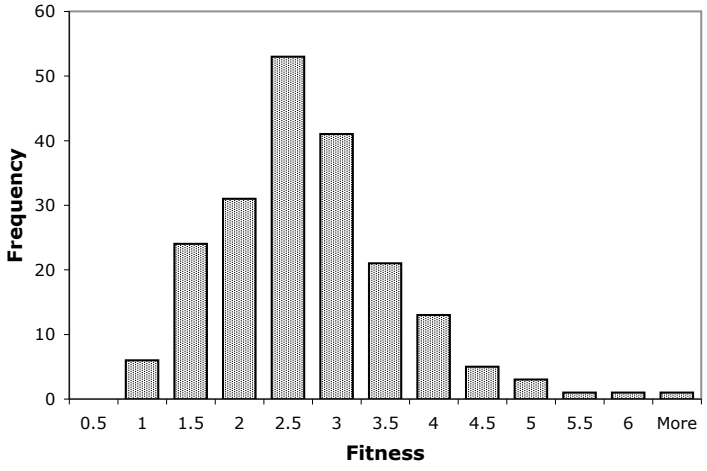
Considering the above recommendations and the shape of the histograms, it is felt that the larger sample size of 200 is adequate to justify the use of the two sample t-test for these samples.



(a) indiscernible ($\delta = -4, r = 5, \eta = -1$)



(b) target ($\delta = -4, r = 2, \eta = -3$)



(c) discernible ($\delta = -4, r = 2, \eta = -2$)

Figure B.6: Histograms Depicting Fitness Distributions

Appendix C

Implementation Details

C.1 Introduction

This Appendix contains implementation details related to the fitness function and GP. Files containing the actual code can be found on the accompanying DVD.

C.2 Fitness Function

Features were determined through C code, R code or by calling methods from the R library. R [52] is an open source system and language which performs statistical computation. R was compiled into a shared library, so that the R code and methods could be readily and efficiently called from C. Table C.1 summarizes the type of code used to calculate each feature. Decomposition of the time series was performed using a combination of R packages, car and stats, along with supplementary R code. The following subsections specify the particular R methods used where applicable.

C.2.1 Non-linearity

Teräsvirta's neural network test [62] was used to quantify non-linearity. The value of the test statistic as calculated by $terasvirta.test$, a method in R package, tseries [21], was selected for the measure. The tseries package depended on R packages, quadprog [65] and zoo [70].

Table C.1: Code Used for Calculating Features

1.	mean	C code
2.	standard deviation	C code
3.	skew	C code
4.	kurtosis	C code
5.	serial correlation	C code
6.	non-linearity	R package: tseries
7.	chaos	C code
8.	self-similarity	R package: fracdiff
9.	periodicity	built-in R package: stats and supplementary R code
10.	mean (tsa)	C code
11.	standard deviation (tsa)	C code
12.	skew (tsa)	C code
13.	kurtosis (tsa)	C code
14.	serial correlation (tsa)	C code
15.	non-linearity (tsa)	R package: tseries
16.	trend	R code
17.	seasonality	R code

C.2.2 Self-similarity

The $fracdiff$ method from the R package, fracdiff [64], was used to obtain the value of d for the Hurst exponent.

C.2.3 Periodicity

The trend was removed from the time series using the $smooth.spline$ method (with $spar = 1$) from R's built-in stats package [52].

The auto correlation function was determined using the acf method from R's built-in stats package [52].

C.2.4 Trend and Seasonally Adjusted Features

Decomposition of the time series was performed with the help of several R methods:

1. Box-Cox transformation used method $box.cox$ from R package, car [20].

2. If periodicity was detected (number of intervals > 1), then decompostion was performed by the STL method using method stl from the built-in R package, stats.

3. If periodicity was not detected, then decomposition was performed by fitting a cubic smoothing spline to extract the seasonal component using method $smooth.spline$ from the built-in R package, stats.

4. The Shapiro-Wilk statistic was obtained using method $shapiro.test$, again from the built-in R package, stats.

C.3 GP

GP runs were performed on Open BEAGLE software [23], a C++, object-oriented, generic framework for performing Evolutionary Computation. It supports tree-based, strongly-typed genetic programming. Aside from defining the function and terminal sets, most of the supplementary code required to customize the system for each problem involved the fitness function. This section first provides implementation details for the symbolic regression experiments, followed by those for the gene gate experiments.

C.3.1 Symbolic Regression Experiments

Function and Terminal Sets

Functions and types pre-defined by Open BEAGLE were used for these experiments.

Fitness Function Code

Calculation of the fitness score for the feature-based fitness function can be divided into 3 steps:

1. **Obtain the "Time Series"**
   Through Open BEAGLE, the expression was directly evaluated as the tree was traversed for the data points within the interval, resulting in an evenly-spaced set of (x, y) values, analogous to a time series. If the experiment involved added noise, it was incorporated in this step.

2. **Determine the Features**
   The subset of features which was chosen for use in the fitness function was then calculated per Section C.2.

3. **Calculate the Fitness Score**
   With the calculated features, the fitness was then determined using equation 5.1 (Section 5.2). Target feature values were determined prior to the run and hard-coded into the program.

C.3.2 Gene Gate Experiments

Functions, Terminals and Types

Functions were set up so that they returned a string which could be concatenated together to form the network expression. Terminals and types built into Open BEAGLE were used as much as possible, though custom types were set up as required to maintain the strong-typing.

Fitness Function Code

Calculation of the fitness score can be divided into 3 steps:

1. **Obtain the Time Series**
   In Open BEAGLE, evaluation of an individual resulted in a string containing the network expression and rates. This string was passed to some C code which constructed an input file for the SPiM simulator. A C system call then invoked SPiM (compiled ocaml code) [49] to perform the simulation. SPiM wrote the simulation results, containing a time course of protein expression levels, to an output file. An example SPiM input file is found in Figure C.1.

2. **Determine the Features**
   For each simulation performed on a candidate expression, C code first read in the time series from the SPiM output file. The subset of features which was chosen for use in the fitness function was then calculated per Section C.2.

3. **Calculate the Fitness Score**
   For each set of calculated features, the fitness was then determined using equation 5.1 and averaged over the specified number of simulations in order to obtain the overall fitness score.

```
directive sample 2000000.0 1000
directive plot !a as "a"
val e=0.1
val d=0.000100
val r=1.000000
val h=0.000100
new a @ r: chan
new b @ r: chan
new c @ r: chan
let tr(b:chan) =
  do !b; tr(b)
  or delay@d
let neg(a:chan, b:chan) =
  do delay@e; (tr(b) | neg(a,b))
  or ?a; neg_(a,b)
and neg_(a:chan,b:chan) = delay@h; neg(a,b)
run(neg(b,c)|neg(c,a)|neg(c,c)|neg(b,a))
```

Figure C.1: Sample SPiM Input File for the Repressilator

Appendix D

Confidence Interval Calculations

D.1 Introduction

Confidence intervals for run success rates were determined using the four plus method which provides accurate results for small samples [47]. The analysis was performed for sample sizes of 10 and 20 at a 95% confidence level.

D.2 Four Plus Confidence Interval Method

The basic method to calculate the confidence interval for a population proportion is accurate only for large samples. Moore [47] recommends using the plus four interval method instead. This approach is appropriate for a confidence level of 90% or higher and a sample size of at least 10.

According to the four plus method, the C% confidence interval for a large population's proportion of successes is:

$$\tilde{p} \pm z^* \sqrt{\frac{\tilde{p}\,(1-\tilde{p})}{n+4}}$$

$$\text{where } \tilde{p} = \frac{\text{count of successes in the sample} + 2}{n+4}$$

n is sample size

z^* is the critical value for the standard Normal density curve with area C between $-z^*$ and z^*

D.3 Results

The 95% confidence intervals ($z^* = 1.960$) for sample sizes of 10 and 20 are found in Tables D.1 and D.2 respectively. Intervals are expressed as a percentage (of success).

Table D.1: 95% Confidence Intervals for a Sample Size of 10

success count	lower bound (% success)	upper bound (% success)
0	0	33
1	0	43
2	5	52
3	11	61
4	17	69
5	24	76
6	31	83
7	39	89
8	48	95
9	57	100
10	67	100

Table D.2: 95% Confidence Intervals for a Sample Size of 20

success count	lower bound (% success)	upper bound (% success)
0	0	19
1	0	26
2	2	32
3	5	37
4	8	42
5	11	47
6	14	52
7	18	57
8	22	61
9	26	66
10	30	70
11	34	74
12	39	78
13	43	82
14	48	86
15	53	89
16	58	92
17	63	95
18	68	98
19	74	100
20	81	100